The Bahá'í Calendar

In the not far distant future it will be necessary that all peoples in the world agree on a common calendar. It seems, therefore, fitting that the new age of unity should have a new calendar free from the objections and associations which make each of the older calendars unacceptable to large sections of the world's population, and it is difficult to see how any other arrangement could exceed in simplicity and convenience that proposed by the Báb.

> John Ebenezer Esslemont: Bahá'u'lláh and the New Era: An Introduction to the Bahá'í Faith (1923)¹

16.1 Structure

The Bahá'í (or Badī') calendar begins its years on the day of the vernal equinox. If the actual time of the equinox in Tehran occurs after sunset, then the year begins a day later [3]. This astronomical version of the Bahá'í calendar [4] is described in Section 16.3. Until recently, practice in the West had been to begin years on March 21 of the Gregorian calendar, regardless. This arithmetical version is described in Section 16.2. The calendar, based on cycles of 19, was established by the Bāb (1819–1850), the martyred forerunner of Bahā'u'llāh, founder of the Bahá'í faith.

As in the Hebrew and Islamic calendars, days are from sunset to sunset. Unlike those calendars, years are solar; they are composed of 19 months of 19 days each with an additional period of 4 or 5 days after the eighteenth month. Until recently, leap years in the Western version of the calendar followed the same pattern as in the Gregorian calendar. As on the Persian calendar, the week begins on Saturday; weekdays have the following names (in Arabic):

Saturday	Jalāl	جلال	(Glory)
Sunday	Jamāl	جمال	(Beauty)
Monday	Kamāl	كمال	(Perfection)
Tuesday	Fiḍāl	فضال	(Grace)
Wednesday	'Idāl	عدال	(Justice)
Thursday	Istijlāl	استجلال	(Majesty)
Friday	Istiqlāl	استقلال	(Independence)

انویسندگان لزوماً با نقطه نظرهای در عبارت موافقت ندارند.

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:38:01, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.019

The months are called

(1) Bahā'	بهاء	(Splendor)	19 days
(2) Jalāl	جلال	(Glory)	19 days
(3) Jamāl	جمال	(Beauty)	19 days
(4) 'Azamat	عظمت	(Grandeur)	19 days
(5) Nūr	نور	(Light)	19 days
(6) Raḥmat	رحمت	(Mercy)	19 days
(7) Kalimāt	كلمات	(Words)	19 days
(8) Kamāl	كمال	(Perfection)	19 days
(9) Asmā'	اسماء	(Names)	19 days
(10) 'Izzat	عرّت	(Might)	19 days
(11) Mashīyyat	مشيئت	(Will)	19 days
(12) 'Ilm	علم	(Knowledge)	19 days
(13) Qudrat	قدرت	(Power)	19 days
(14) Qawl	قَول	(Speech)	19 days
(15) Masā'il	مسائل	(Questions)	19 days
(16) Sharaf	شرف	(Honor)	19 days
(17) Sulțān	سلطان	(Sovereignty)	19 days
(18) Mulk	مُلک	(Dominion)	19 days
Ayyām-i-Hā	اتيام ها	(Days of God)	4 {5} days
(19) 'Alā'	علاًء	(Loftiness)	19 days

The leap-year variation is given in braces. The 19 days of each month have the same names as the months, except that there is no intercalary Ayyām-i-Hā.

Years are also named in a 19-year cycle, called *Vāhid*, meaning "unity" and having a numerological value of 19 in Arabic letters:

(1) Alif	الف	(letter A)
(2) Bā'	باء	(letter B)
(3) Ab	أب	(Father)
(4) Dāl	دال	(letter D)
(5) Bāb	باب	(Gate)
(6) Vāv	واو	(letter V)
(7) Abad	ابد	(Eternity)
(8) Jād	جاد	(Generosity)
(9) Bahā'	بهاء	(Splendor)
(10) Hubb	حُبّ	(Love)
(11) Bahhāj	بہتاج	(Delightful)
(12) Javāb	جوآب	(Answer)
(13) Aḥad	احد	(Single)
(14) Vahhāb	وهاب	(Bountiful)

(15) Vidād	وداد	(Affection)
(16) Badī'	ىدب	(Beginning)
(17) Bahī	بهسى	(Luminous)
(18) Abhā	ابہتی	(Most Luminous)
(19) Vāhid	واحد	(Unity)

There is also a 361-year major cycle, called *Kull-i-Shay* (the name has the numerological value $361 = 19^2$ in Arabic). Thus, for example, Monday, April 21, 1930 would be called "Kamāl (Monday), the day of Qudrat (the thirteenth), of the month of Jalāl, of the year Bahhāj (the eleventh), of the fifth Vāḥid, of the first Kull-i-Shay, of the Bahá'í Era."

Accordingly, we represent a Bahá'í date by a list

major cycle year month day

The first component, *major*, is an integer (positive for real Bahá'í dates); the components *cycle*, *year*, and *day*, take on integer values in the range 1 . . 19; because the intercalary period interrupts the sequence of month numbers, *month* is either an integer between 1 and 19 or else the special constant value

$$ayyam-i-ha \stackrel{\text{def}}{=} 0 \tag{16.1}$$

The epoch of the calendar, day 1 of year 1 B.E.,² is March 21, 1844 (Gregorian):

bahai-epoch
$$\stackrel{\text{def}}{=}$$
 fixed-from-gregorian $\left(\begin{array}{c|c} 1844 & \text{march} & 21 \end{array} \right)$ (16.2)

which is R.D. 673222.

16.2 The Arithmetical Calendar

Mr. Frank *E.* Osborne read a complete Bahai calendar on which he has been working for the past four or five years. Abdul-Baha gave it his verbal sanction. It was referred to the executive board.

Star of the West, vol. 8 (1917)

The Bahá'í calendar used in the West until 2015 was based on the Gregorian calendar, and thus its functions are relatively straightforward:



² Bahá'í Era

where

$$g-year = 361 \times (major - 1) + 19 \times (cycle - 1) + year - 1$$

+ gregorian-year-from-fixed (bahai-epoch)

We first find the corresponding Gregorian year by counting how many years (361 for each major cycle and 19 for each minor cycle) have elapsed since the epoch in 1844. Starting with the R.D. date of the last day (March 20) of the prior Bahá'í year, we add the number of days in the given month plus 19 days for each month, except that the intercalary period has only 4 or 5 days (for a total of 346 or 347 days), depending on whether February of the Gregorian calendar had a leap day or not.

The inverse function is

bahai-from-fixed (*date*)
$$\stackrel{\text{def}}{=}$$
 (16.4)

major cycle year month day

where

 $g-year = \operatorname{gregorian-year-from-fixed} (date)$ $start = \operatorname{gregorian-year-from-fixed} (bahai-epoch)$ years = g-year - start $- \left\{ \begin{array}{c} 1 & \operatorname{if} date \leq \operatorname{fixed-from-gregorian} \\ \left(\boxed{g-year} \operatorname{march} 20 \right) \\ 0 & \operatorname{otherwise} \end{array} \right\}$ $major = \left\lfloor \frac{years}{361} \right\rfloor + 1$ $cycle = \left\lfloor \frac{1}{19} \times (years \mod 361) \right\rfloor + 1$ $year = (years \mod 19) + 1$ $days = date - \operatorname{fixed-from-bahai} \\ \left(\boxed{major} \boxed{cycle} \boxed{year} \boxed{11} \right)$ $month = \left\{ \begin{array}{c} 19 & \operatorname{if} date \geq \operatorname{fixed-from-bahai} \\ \left(\boxed{major} \boxed{cycle} \boxed{year} \boxed{19} \end{aligned} \right)$ $\frac{date}{19} = \operatorname{fixed-from-bahai} \\ \left(\boxed{major} \boxed{cycle} \boxed{year} \boxed{19} \right) \\ \operatorname{ayyam-i-ha} \\ \operatorname{if} date \geq \operatorname{fixed-from-bahai} \\ \left(\boxed{major} \boxed{cycle} \boxed{year} \boxed{19} \right) \\ \left\lfloor \frac{days}{19} \right\rfloor + 1 \\ \operatorname{otherwise} \end{array} \right)$

$$day = date + 1$$

- fixed-from-bahai ($major | cycle | year | month | 1$)

Here we compute the number of years that have elapsed since the start of the Bahá'í calendar by looking at the Gregorian year number, considering whether the date is before or after Bahá'í New Year, and then using the result to get the number of elapsed major and minor cycles and years within the cycle. Division of the remaining days by 19, the length of a month, gives the month number, but again special consideration must be given for the intercalary period and for the last month of the Bahá'í year.

16.3 The Astronomical Calendar

The chief element of the day after to-morrow in the political calendar will be All Europe as One. There can be no doubt on this point. Unhappily, however, no European nation seems yet to have realized the fact.

German contributor to *Revue de Genéve*, quoted in *The Literary Digest*, vol. 75 (1922)

The Bahá'í year was intended [3] to begin at the sunset preceding the vernal equinox, which is frequently a day before or after March 21. The location at which sunset occurs for this purpose had been undetermined for some time, as explained in the following explanatory letter [2] written in 1974:

Until the Universal House of Justice decides upon the spot on which the calculations for establishing the date of Naw-Rúz each year are to be based it is not possible to state exactly the correspondence between Bahá'í dates and Gregorian dates for any year. Therefore for the present the believers in the West commemorate Bahá'í events on their traditional Gregorian anniversaries. Once the necessary legislation to determine Naw-Rúz has been made, the correspondence between Bahá'í and Gregorian dates will vary from year to year depending upon whether the Spring Equinox falls on the 20th, 21st or 22nd of March. In fact in Persia the friends have been, over the years, following the Spring Equinox as observed in Tehran, to determine Naw-Rúz, and the National Spiritual Assembly has to issue every year a Bahá'í calendar for the guidance of the friends. The Universal House of Justice feels that this is not a matter of urgency and, in the meantime, is having research conducted into such questions.

Thus, the version of the Bahá'í calendar employed in the Near East (which included, besides Iran, also Israel, Persian Gulf countries, and the Arabian Peninsula) used Tehran for determining the time of sunset on the day of the equinox, which in turn fixes the first day of the year. In 2014, the decision was taken to use Tehran as the determining location the world over [4]:

"The Festival of Naw-Rúz falleth on the day that the sun entereth the sign of Aries," Bahá'u'lláh explains in His Most Holy Book, "even should this occur no more than one minute before sunset." However, details have, until now, been left undefined. We have decided that Tihrán, the birthplace of the Abhá Beauty, will be the spot on the earth that will serve as the standard for determining, by means of astronomical computations from reliable sources, the moment of the vernal equinox in the northern hemisphere and thereby the day of Naw-Rúz for the Bahá'í world.

This change took effect with the year that began on March 21, 2015.

For fixing the time of sunset in Tehran, these coordinates are used:³

bahai-location
$$\stackrel{\text{def}}{=}$$
 (16.5)

 35.696111° 51.423056° 0 m $3\frac{1}{2}^{\text{h}}$

The determination of the U.T. moment of sunset on any specified day is straightforward:

bahai-sunset (*date*)
$$\stackrel{\text{def}}{=}$$
 (16.6)

universal-from-standard (sunset (*date*, bahai-location), bahai-location)

The first day of the year on the new, astronomical, Bahá'í calendar is the day on which the vernal equinox occurs before sunset. To implement the astronomical form of the calendar, we imitate the method used for the astronomical Persian calendar in Section 15.2. The date of the new year is computed using formula (14.43), analogously to what was done for the Persian calendar (page 259), by beginning shortly before the equinox and searching for the sunset when the longitude of the sun first switches from large (close to 360°) to small (less than 2°):

astro-bahai-new-year-on-or-before
$$(date) \stackrel{\text{def}}{=} (16.7)$$

$$\underset{day \ge \lfloor approx \rfloor - 1}{\text{MIN}} \left\{ \text{solar-longitude (bahai-sunset (day))} \le \text{spring} + 2^{\circ} \right\}$$

where

approx = estimate-prior-solar-longitude (spring, bahai-sunset (*date*))

Because of the unequal distribution of leap years on the Gregorian calendar, the equinox will be as early as 5:27 p.m. in Tehran on March 19 in 2096, which is before sunset, and it was as late as 10:41 p.m. on March 21 in 1903, long after sunset. By the new rule, the year would begin on March 19 in the former case and March 22 in the latter.

³ The elevation of Tehran (1180 m) is not taken into account in the sunset calculation, because the mountains to its west are at about the same height, so apparent sunset occurs at approximately the same time as astronomical sunset at zero elevation [1].

1 0

To convert a Bahá'í date on the new calendar into a fixed date, we take the R.D. date of the Bahá'í New Year and add 19 days for each full month plus the number of elapsed days in the current month. The intercalary days and last month of the year must be treated as exceptions: days in Ayyām-i-Hā are preceded by 18 full months (that is, 342 days); because the length of that period differs in ordinary and leap years, for dates in the last month, we count backwards from the following New Year. In the following function, we multiply the number of years since the epoch by the mean tropical year length, plus or minus half a year, and then use **astro-bahai-new-year-on-or-before** to get the R.D. date of the subsequent or prior Bahá'í New Year:

where

$$years = 361 \times (major - 1) + 19 \times (cycle - 1) + year$$

The inverse function is

astro-bahai-from-fixed (date)
$$\stackrel{\text{def}}{=}$$
(16.9)majorcycleyearmonthday

where

1

$$new-year = astro-bahai-new-year-on-or-before (date)$$

$$years = round \left(\frac{new-year - bahai-epoch}{mean-tropical-year} \right)$$

$$major = \left\lfloor \frac{years}{361} \right\rfloor + 1$$

$$cycle = \left\lfloor \frac{1}{19} \times (years \mod 361) \right\rfloor + 1$$

$$year = (years \mod 19) + 1$$

$$days = date - new \cdot year$$

$$\begin{cases} 19 \quad \text{if } date \\ \geqslant \text{fixed-from-astro-bahai} \\ \left(\boxed{major \ cycle \ year \ 19 \ 1} \right) \\ \text{ayyam-i-ha} \\ \text{if } date \\ \geqslant \text{fixed-from-astro-bahai} \\ \left(\boxed{major \ cycle \ year \ ayyam-i-ha \ 1} \right) \\ \left\lfloor \frac{days}{19} \right\rfloor + 1 \\ \text{otherwise} \end{cases}$$

$$day = date + 1$$

$$- \text{fixed-from-astro-bahai} \\ \left(\boxed{major \ cycle \ year \ month \ 1} \right)$$

Here we compute the number of years that have elapsed since the start of the Bahá'í calendar by dividing the numbers of days since the epoch by the mean tropical year length and then using the result to get the number of elapsed major and minor cycles and years within the cycle. Division of the remaining days by 19 (the length of a Bahá'í month) gives the month number but, again, consideration must be given to the intercalary days and for the last month of the Bahá'í year.⁴

⁴ The published tables of the ad hoc calendar committee at the Bahá'í World Centre for the years 172–221 B.E. (2015–2064 c.E.) were prepared "using data provided by Her Majesty's Nautical Almanac Office in the United Kingdom" and are available at wilmetteinstitute.org/wp-content/uploads/2014/11/Bahai-Dates-172-to-221-B-E-_UK-December-2014.pdf. There is almost complete correspondence between the dates calculated with our functions and those in the table. The only divergence is for 2026, for which the table has New Year occurring on March 21, and our calculations place it on the previous day. This is, however, a very close call, since both sunset and the equinox will occur on March 20 between 6:15 and 6:16 p.m. local standard time in Tehran. On account of the very close proximity of the two events, the decision was made to set Bahá'í New Year to be March 21 [1].

16.4 Holidays and Observances

In all, there are 58 excusable days of observance for various religions on the state's academic calendar – which requires schools to open 180 days a year. It should be noted that in many schools, Christmas and Hanukkah are losing ground to Birth of the Bab Day (Baha'i), and the Rama Navami (Hindu) and Eid El Fitr (Islamic) holy days.

Lisa Suhay: "Want the Day Off? Get Some Religion," The New York Times (September 19, 1999)

(16.10)

When the Bahá'í calendar used in the West was synchronized with the Gregorian, holidays were a trivial matter. Bahá'í New Year was always celebrated on March 21, the assumed date of the spring equinox. It is called the Feast of Naw-Rūz, like the Persian New Year, which also celebrates the vernal equinox (see Chapter 15). The computation is trivial:

bahai-new-year
$$(g-year) \stackrel{\text{def}}{=}$$

The only holiday that was not aligned with the Gregorian calendar was Ayyāmi-Hā 4, which fell on March 1 in ordinary years, but on February 29 in leap years.

march

21

The other major holidays are the Birth of the Bāb (which was celebrated on 'Ilm 5 = October 20), the Birth of Bahā'u'llāh (which was celebrated on Qudrat 9 = November 12), the Feast of Ridvān (Jalāl 13 = April 21 with the old Western version), Ridvān 9 (Jamāl 2 = April 29), Ridvān 12 (Jāmal 5 = May 2), the Declaration of the Bāb ('Aẓamat 8 = May 24), the Ascension of Bahā'u'llāh ('Aẓamat 13 = May 29), and the Martyrdom of the Bāb (Raḥmat 17 = July 10). Two other obligatory observances are the Day of the Covenant (Qawl 4 = November 26) and the Ascension of 'Abdu'l-Bahā (Qawl 6 = November 28). There are additional days of significance, including the first day of each month (known as the Nineteen Day Feast) and the whole last month (comprising fast days).

With the new calendar, which depends on the actual time at which the equinox occurs, Bahá'í Naw-Rūz, on Bahá 1, coincides with Persian Nowruz unless the equinox occurs between noon and sunset in Tehran. A straightforward way to determine the date of Bahá'í Naw-Rūz is as follows:

$$\mathbf{naw-ruz}(g-year) \stackrel{\text{def}}{=}$$
(16.11)

astro-bahai-new-year-on-or-before (gregorian-new-year (g-year + 1))

Determining the date of holidays, apart from Birth of the Bāb and the Birth of Bahā'u'llāh, on the new astronomical calendar (as previously for the Eastern version) is simply a matter of counting a fixed number of days from Naw-Rūz, or before Naw-Rūz in the case of the month of 'Alā'. For example, we have

feast-of-ridvan
$$(g-year) \stackrel{\text{def}}{=} \mathbf{naw-ruz} (g-year) + 31$$
 (16.12)

The other major holidays on the Bahá'í calendar are also observed on Bahá'í dates (given above), except for four that had been linked in the East to the Islamic calendar (Chapter 7) instead: Declaration of the Bāb (Islamic date Jamādā I 5), Martyrdom of the Bāb (Sh'abān 28), Birth of the Bāb (Muḥarram 1), and the Birth

of Bahā'u'llāh (Muharram 2). (At the Bahá'í World Centre in Israel, these four had been observed on their Islamic dates whereas the other holidays had been observed on their Gregorian dates.) Following the recent decision Bahá'í dates are to be used for the first two of these four holidays, Declaration of the Bāb (on 'Azamat 8) and Martyrdom of the Bāb (Raḥmat 17), while astronomical lunisolar dates are to be used everywhere for the other two: The rule is that the Birth of the Bāb and the Birth of Bahā'u'llāh are observed on the first and second day, respectively, of the eighth lunisolar month, counting new moons from sunset at the end of Naw-Rūz. Using **new-moon-at-or-after** (page 231) for this purpose, we have:

birth-of-the-bab
$$(g-year) \stackrel{\text{def}}{=} \begin{cases} day + 1 & \text{if } m_8 < set_8 \\ day + 2 & \text{otherwise} \end{cases}$$
 (16.13)

where

ny = naw-ruz (g-year) $set_1 = bahai-sunset (ny)$ $m_1 = new-moon-at-or-after (set_1)$ $m_8 = new-moon-at-or-after (m_1 + 190)$ $day = fixed-from-moment (m_8)$ $set_8 = bahai-sunset (day)$

and m_8 is the moment of the eighth new moon of the year. If new moon is before sunset, then the eighth month begins at sunset; if the new moon is after sunset, the month begins one day later.

References

- [1] Email communication from the Bahá'í World Centre, July 22, 2015.
- [2] Letter written on behalf of the Universal House of Justice to the National Spiritual Assembly of the Bahá'í of the United States, October 30, 1974.
- [3] Universal House of Justice, *The Bahá'í World: An International Record*, vol. xviii, Bahá'í World Center, Haifa, pp. 598–601, 1986.
- [4] Universal House of Justice, "Regarding the implementation of the Badí' calendar," July 10, 2014. Available at universalhouseofjustice.bahai. org/activities-bahai-community/20140710_001.

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:38:01, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.019



Print of the French Revolutionary calendar month of Vendémiaire by Laurent Guyot, after Jean-Jacques Lagrenée, the younger, Paris. (Courtesy of Bibliothèque Nationale de France, Paris.)

Appendix D

Lisp Implementation

It has been often said that a person does not really understand something until he teaches it to someone else. Actually a person does not really understand something until he can teach it to a computer, i.e., express it as an algorithm. Donald E. Knuth: "Computer Science and its Relation to Mathematics," American Mathematical Monthly (1974)

This appendix contains the complete Common Lisp implementation of the calendar functions described in the main text; the equation numbers given here are those of the corresponding functions in the text. Some Lisp functions have no corresponding equation in the text—these are constructors, selectors, and standard mathematical operations that are also used to control the typesetting: the functions in the main text were automatically typeset from the definitions in this appendix. The Lisp functions are available over the World Wide Web at

www.cambridge.org/calendricalcalculations

Please bear in mind the limits of the License and that the copyright on this book includes the code. Also please keep in mind that if the result of any calculation is critical, it should be verified by independent means.

For licensing information about nonpersonal and other uses, contact the authors. The code is distributed in the hope that it may be useful but without any warranty as to the accuracy of its output and with liability limited to return of the price of this book, which restrictions are set forth on page xli.

D.1 Basics

D.1.1 Lisp Preliminaries

For readers unfamiliar with Lisp, this section provides the bare necessities. A complete description can be found in [2].

All functions in Lisp are written in prefix notation. If f is a defined function, then

```
(f e0 e1 e2 ... en)
```

applies f to the n + 1 arguments e0, e1, e2, ..., en. Thus, for example, + adds up a list of numbers; for example,

(+ 1 -2 3)

adds the three numbers and returns the value 2. The Lisp functions $-, \star,$ and / work similarly, to subtract, multiply, and divide, respectively, a list of numbers. In a similar fashion, <= (\leq) checks that the numbers are in nondecreasing order and yields true (t in Lisp) if the relations hold. For instance,

(<= 1 2 3)

evaluates to t. The Lisp functions =, / = (not equal), <, >, and >= (greater than or equal) are similar. The predicate evenp tests whether an integer is even.

Lists are Lisp's main data structure. To construct a list (e0 e1 e2 ... en) the expression

(list e0 e1 e2 ... en)

is used. The function nth, used as (nth i l), extracts the *i*th element of the list l, indexing from 0; the predicate member, used as (member x l), tests whether x is an element of l. To get the first (indexed 0), second, and so on, through tenth elements of a list, we use the functions first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, and tenth. The tail of the list, consisting of all the elements but the first, is obtained using rest. The empty list is represented by nil.

Constants are defined with the defconstant command, which has the syntax

(defconstant constant-name expression)

For example,

1

2

3

4

(defconstant sunday	(1.53)
;; TYPE day-of-week	
:: Residue class for Sunday.	

0)

1 2 3 4	(defconstant monday ;; TYPE day-of-week ;; Residue class for Monday. 1)	(1.54)	Notice that semicolons mark the start of comments. "Type" information is given in comments each function. Although Common Lisp has its own system of type declarations, we prefered the simp untyped, Lisp, but we annotate each function and constant to aid the reader in translating our code a typed language. The base types are defined in Table A.1, beginning on page 389. To distinguish in the code between empty lists (nil) and the truth value "false," we define	i for pler, into
1 2 3 4	(defconstant tuesday ;; TYPE day-of-week ;; Residue class for Tuesday. 2)	(1.55)	<pre>1 (defconstant false 2 ;; TYPE boolean 3 ;; Constant representing false. 4 nil) For "true," we define</pre>	
1 2 3 4	(defconstant wednesday ;; TYPE day-of-week ;; Residue class for Wednesday. 3)	(1.56)	<pre>1 (defconstant true 2 ;; TYPE boolean 3 ;; Constant representing true. 4 t)</pre>	
1 2 3 4	(defconstant thursday ;; TYPE day-of-week ;; Residue class for Thursday. 4)	(1.57)	<pre>We also use a string constant to signify an error value: 1 (defconstant bogus (1 2 ;; TYPE string 3 ;; Used to denote nonexistent dates. 4 "bogus")</pre>	.97)
1 2 3 4	(defconstant friday ;; TYPE day-of-week ;; Residue class for Friday. 5)	(1.58)	The function equal can be used to check lists and strings for equality. Functions are defined using the defun command, which has the following syntax: (defun function-name (param1 paramn) expression) For example, we compute the day of the week of an R.D. date (page 33) with	
1 2 3 4	(defconstant saturday ;; TYPE day-of-week ;; Residue class for Saturday. 6)	(1.59)	1 (defun day-of-week-from-fixed (date) (1 2 ;; TYPE fixed-date -> day-of-week (1 3 ;; The residue class of the day of the week of date. (1 4 (mod (- date (rd 0) sunday) 7)) (1 and we implement julian day calculations by writing (1	.60)

D.1 Basics 471

1	(defconstant jd-epoch	(1.3)	1	(defun fixed-from-mjd (mjd)	(1.7)
2	;; TYPE moment		2	;; TYPE julian-day-number -> fixed-date	
3	;; Fixed time of start of the julian day number.		3	;; Fixed date of modified julian day number mjd.	
4	(rd -1721424.5L0))		4	(+ mjd mjd-epoch))	
Com	mon Lisp uses L0 after a number to specify unscaled maximum-precision (at least 50-bit) cor	istants.		(define mid from fixed (data)	(1.8)
	We use the identity function		1	(delun mjd-from-fixed (date)	(1.8)
			2	;; TYPE lixed-date -> Julian-day-humber	
1	(defun rd (tee)	(1.1)	3	(data mid enoch))	
2	;; TYPE moment -> moment		4	(- date mjd-epotn))	
3	;; Identity function for fixed dates/moments. If internal				
4	;; timekeeping is shifted, change epoch to be RD date of		1	(defconstant unix-epoch	(1.9)
5	;; origin of internal count. epoch should be an integer.		2	;; TYPE fixed-date	
6	(let* ((epoch 0))		3	;; Fixed date of the start of the Unix second count.	
7	(- tee epoch)))		4	(rd 719163))	
to ma	ake it easy to adapt the code to an alternate fixed-date enumeration—all that is needed is to	change	1	(defun moment-from-unix (s)	(1.10)
the v	alue of epoch in line 6 of rd. The Common Lisp construct let* defines a sequence of co	nstants	2	:: TYPE second -> moment	()
(poss	sibly in terms of previously defined constants) and ends with an expression whose value is re	eturned	3	:: Fixed date from Unix second count s	
by th	e construct.		4	(+ unix-epoch (/ s 24 60 60)))	
1	(defun moment-from-jd (jd)	(1.4)			
2	;; TYPE julian-day-number -> moment		1	(defun unix-from-moment (tee)	(1.11)
3	;; Moment of julian day number jd.		2	;; TYPE moment -> second	
4	(+ jd jd-epoch))		3	;; Unix second count from moment tee	
			4	(* 24 60 60 (- tee unix-epoch)))	
1	(defun jd-from-moment (tee)	(1.5)			
2	;; TYPE moment -> julian-day-number		1	(defun fixed-from-jd (jd)	(1.13)
3	;; Julian day number of moment tee.		2	;; TYPE julian-day-number -> fixed-date	
4	(- tee jd-epoch))		3	;; Fixed date of julian day number jd.	
			4	(floor (moment-from-jd jd)))	
1	(defconstant mjd-epoch	(1.6)			/4 · · ·
2	;; TYPE fixed-date		1	(defun jd-from-fixed (date)	(1.14)
3	;; Fixed time of start of the modified julian day number.		2	;; TYPE fixed-date -> julian-day-number	
4	(rd 678576))		3	;; Julian day number of fixed date.	
			4	(jd-irom-moment date))	

As another example of a function definition, we can define a function (inconveniently named floor in Common Lisp) to return the (truncated) integer quotient of two integers, |m/n|:

(defun quotient (m n) 1

- ;; TYPE (real nonzero-real) -> integer 2
- 3 ;; Whole part of m/n.
- 4 (floor m n))

The floor function can also be called with one argument. Thus

(floor x)

is |x|, the greatest integer less than or equal to x.

As a final example of function definitions, note that the Common Lisp function mod always returns a nonnegative value for a positive divisor; we use this property occasionally, but we also need a function like mod with its values adjusted in such a way that the modulus of a multiple of the divisor is the divisor itself rather than 0. To define this function, we write

1	(defun amod (x y)	(1.29)
2	;; TYPE (integer nonzero-integer) -> integer	
3	;; The value of $(x \mod y)$ with y instead of 0.	
4	(+ y (mod x (- y))))	

This is typeset as $x \mod [1 \dots y]$ in the main text.

More generally, we use a function that shifts the modulus into a specified range of values [1]:

1	(defun mod3 (x a b)	
2	;; TYPE (real real real) -> real	
3	;; The value of x shifted into the range	
4	;; $[ab)$. Returns x if $a=b$.	
5	(if (= a b)	
6	х	
7	$(+ a \pmod{(-x a)} (- b a))))$	
This is	s typeset as $x \mod [a \dots b]$; see page 22.	

The function if has three arguments: a boolean condition, a then-expression, and an elseexpression. The cond statement, also used in what follows, lists a sequence of tests and values and serves as a generalized case statement.

For convenience in expressing our calendar functions in Lisp, we introduce a macro to compute sums. The expression

(sum fikp)

computes

$$\sum_{k \leq i < \min_{j \geq k} \{\neg p(j)\}} f(i);$$

that is, the expression f(i) is summed for all $i = k, k + 1, \ldots$, continuing only as long as the condition p(i) holds. The sum is 0 if p(k) is false. Our Common Lisp definition of sum uses the versatile loop construct and is as follows:

(defmacro sum (expression index initial condition) (1.30)1 2 ;; TYPE ((integer->real) * integer (integer->boolean)) 3 ;; TYPE -> real 4 ;; Sum expression for index = initial and successive ;; integers, as long as condition holds. 5 '(loop for ,index from ,initial 6 7 while , condition sum ,expression)) 8

This is the first of the few instances in which we use macros and not functions; it allows us to avoid the issue of passing functions to functions.

A similar macro, prod, is used for products:

mod3 (x a b)	(1.24)	1	(defmacro prod (expression index initial condition)	(1.31)
PE (real real) -> real		2	;; TYPE ((integer->real) * integer (integer->boolean))	
e value of x shifted into the range		3	;; TYPE -> real	
b). Returns x if a=b.		4	;; Product of expression for index = initial and successive	
= a b)		5	;; integers, as long as condition holds.	
		6	'(apply '*	
a (mod (- x a) (- b a)))))		7	(loop for ,index from ,initial	
		8	while , condition	
$x \mod [a \dots b)$; see page 22.		9	collect ,expression)))	

The c	ollect construct gathers a list of factors and the function apply applies the multiplication operation to	1	(defmacro final (index initial condition)	(1.33)
that li	st.	2	;; TYPE (* integer (integer->boolean)) -> integer	
	A summation macro sigma and a summation function poly for polynomials are used mainly in the	3	;; Last integer greater or equal to initial such that	
astron	iomical code:	4	;; condition holds.	
		5	'(loop for ,index from ,initial	
1	(defmacro sigma (list body)	6	when (not , condition)	
2	;; TYPE (list-of-pairs (list-of-reals->real))	7	<pre>return (1- ,index)))</pre>	
3	;; TYPE -> real			
4	;; list is of the form ((i1 l1)(in ln)).	The f	unction 1- decrements a number by one; the similar function 1+ increments by one.	
5	;; Sum of body for indices i1in		We also use binary search—see equation (1.35)—expressed as the macro binary-search :	
6	;; running simultaneously thru lists l1ln.			
7	'(apply '+ (mapcar (function (lambda	1	(defmacro binary-search (] lo h hi x test end)	(1.35)
8	,(mapcar 'car list)	2	:: TYPE (* real * real * (real->boolean)	(1100)
9	, body))	3	:: TYPE ((real real)->boolean)) -> real	
10	,@(mapcar 'cadr list))))	4	:: Bisection search for x in [lo., bi] such that	
		5	;; end holds, test determines when to go left.	
		6	(let* ((left (gensym)))	
1	(defun poly (x a)	7	'(do* ((.x false (/ (+ .h .l) 2))	
2	;; TYPE (real list-of-reals) -> real	8	(.left_falsetest)	
3	;; Sum powers of x with coefficients (from order 0 up)	9	(1 lo (if left 1 x))	
4	;; in list a.	10	(h hi (if left x h)))	
5	(if (equal a nil)	11	(m, m) (11 (11 (11 (m)))	
6	0			
7	(+ (first a) (* x (poly x (rest a))))))	The c	operation of loop	
		The e	Binary search is used mainly for function inversion:	
	The function mapcar applies a function (expressed by means of function and lambda) to each		binary search is used manny for function inversion.	
eleme	nt of a list.			(1.20)
	Two additional sum-like macros are used for searching; the first implements the MIN function, equation	1	(defmacro invert-angular (f y r)	(1.36)
(1.32)), and the second implements MAX, equation (1.33):	2	;; TYPE (real->angle real interval) -> real	
		3	;; Use disection to find inverse of angular function	
1	(defmacro next (index initial condition) (1.32)	4	;; f at y within interval r.	
2	;; TYPE (* integer (integer->boolean)) -> integer	5	(let* ((varepsilon 1/100000)); Desired accuracy	
3	;; First integer greater or equal to <i>initial</i> such that	6	'(binary-search 1 (begin ,r) u (end ,r) x	
4	;; condition holds.	7	(< (mod (- (, f x), y) 360) (deg 180))	
5	`(loop for ,index from ,initial	8	<pre>(< (- u 1) ,varepsilon))))</pre>	
6	when , condition			
7	return ,index))	The in	nterval selectors, begin and end , are defined below.	

D.1.2 Basic Code	1	(defun seconds (clock)	
To extract a particular component from a date, we use, when necessary, the functions standard-month, standard-day, and standard-year. For example:	2 3	;; TYPE clock-time -> second (third clock))	
<pre>1 (defun standard-month (date) 2 ;; TYPE standard-date -> standard-month 3 ;; Month field of date = (year month day). 4 (second date))</pre>	1 2 3	(defun time-of-day (hour minute second) ;; TYPE (hour minute second) -> clock-time (list hour minute second))	
<pre>1 (defun standard-day (date) 2 ;; TYPE standard-date -> standard-day 3 ;; Day field of date = (year month day).</pre>	1 2 3 4	<pre>(defun fixed-from-moment (tee) ;; TYPE moment -> fixed-date ;; Fixed-date from moment tee. (floor tee))</pre>	(1.12)
4 (third date))	1 2 3	(defun sign (y) ;; TYPE real -> {-1,0,+1} ;; Sign of y.	(1.16)
1 (defun standard-year (date)	4	(cond	
2 ;; TYPE standard-date -> standard-year	5	((< y 0) -1)	
<pre>3 ;; Year field of date = (year month day). 4 (first date))</pre>	6 7	((> y 0) +1) (t 0)))	
Such constructors and selectors could be defined as macros or Lisp structures. In languages like C or C++, these would more naturally be field selection in fixed-length records rather than lists. We also have	1 2 3 4	<pre>(defun time-from-moment (tee) ;; TYPE moment -> time ;; Time from moment tee. (mod tee 1))</pre>	(1.18)
1 (defun hour (clock)			
<pre>2 ;; TYPE clock-time -> hour 3 (first clock))</pre>	1 2 3	<pre>(defun list-of-fixed-from-moments (ell) ;; TYPE list-of-moments -> list-of-fixed-dates ;; List of fixed dates corresponding to list ell</pre>	(1.37)
<pre>1 (defun minute (clock) 2 ;; TYPE clock-time -> minute 3 (second clock))</pre>	4 5 6 7	<pre>;; of moments. (if (equal ell nil) nil (append (list (fixed-from-moment (first ell))) </pre>	
	0	(iist-of-lixed-from-moments (rest ell)))))	

1	(defun interval (t0 t1)		1	(defun positions-in-range (p c cap-Delta range)	(1.40)
2	;; TYPE (moment moment) -> interval		2	;; TYPE (nonegative-real positive-real	
3	;; Half-open interval [t0t1).		3	;; TYPE nonegative-real interval) -> list-of-moments	
4	(list t0 t1))		4	;; List of occurrences of moment p of c-day cycle	
			5	;; within range.	
1	(defun interval-closed (t0 t1)		6	;; cap-Delta is position in cycle of RD moment 0.	
2	;; TYPE (moment moment) -> interval		7	(let* ((a (begin range))	
3	;; Closed interval [t0t1].		8	(b (end range))	
4	(list t0 t1))		9	(date (mod3 (- p cap-Delta) a (+ a c))))	
			10	(if (>= date b)	
1	(defun begin (range)		11	nil	
2	;; TYPE interval -> moment		12	(append (list date)	
3	;; Start t0 of range [t0t1) or [t0t1].		13	(positions-in-range p c cap-Delta	
4	(first range))		14	(interval (+ a c) b))))))	
1	(defun end (range)				
2	;; TYPE interval -> moment			The following two functions for mixed radiu conversions (see Section 1.10) take on anti-	anal thind
3	;; End t1 of range [t0t1) or [t0t1].			The following two functions for mixed-radix conversions (see Section 1.10) take an opti-	onai unru
4	(second range))		paran	eter for the fractional part of the basis:	
1	(defun in-range? (tee range)	(1.38)			
2	;; TYPE (moment interval) -> boolean		1	(defun from-radix (a b &optional c)	(1.41)
3	;; True if tee is in half-open range.		2	;; TYPE (list-of-reals list-of-rationals list-of-rationals)	
4	(and (<= (begin range) tee) (< tee (end range))))		3	;; TYPE -> real	
			4	;; The number corresponding to a in radix notation	
1	(defun list-range (ell range)	(1.39)	5	;; with base b for whole part and c for fraction.	
2	;; TYPE (list-of-moments interval) -> list-of-moments		6	(/ (sum (* (nth i a)	
3	;; Those moments in list ell that occur in range.		7	(prod (nth j (append b c))	
4	(if (equal ell nil)		8	j i (< j (+ (length b) (length c)))))	
5	nil		9	i 0 (< i (length a)))	
6	(let* ((r (list-range (rest ell) range)))		10	(apply '* c)))	
7	(if (in-range? (first ell) range)				
8	(append (list (first ell)) r)				
9	r))))		where	length measures the length of a list; and	

1	(defun to-radix (x b &optional c)	(1.42)		D.1.3 The Egyptian and Armenian Calendars	
2	;; TYPE (real list-of-rationals list-of-rationals)				
3	;; TYPE -> list-of-reals		1	(defun egyptian-date (year month day)	
4	;; The radix notation corresponding to x		2	;; TYPE (egyptian-year egyptian-month egyptian-day)	
5	;; with base b for whole part and c for fraction.		3	;; TYPE -> egyptian-date	
6	(if (null c)		4	(list year month day))	
7	(if (null b)				
8	(list x)				
9	(append (to-radix (quotient x (nth (1- (length b)) b))		1	(defconstant egyptian-epoch	(1.46)
10	(butlast b) nil)		2	;; TYPE fixed-date	
11	(list (mod x (nth (1- (length b)) b)))))		3	;; Fixed date of start of the Egyptian (Nabonasser)	
12	(to-radix (* x (apply '* c)) (append b c))))		4	;; calendar.	
			5	;; JD 1448638 = February 26, 747 BCE (Julian).	
which	is implemented recursively.		6	(fixed-from-jd 1448638))	
1	(defun time-from-clock (hms)	(1.43)			
2	;; TYPE clock-time -> time		1	(defun fixed-from-egyptian (e-date)	(1.47)
3	;; Time of day from hms = hour:minute:second.		2	;; TYPE egyptian-date -> fixed-date	
4	(/ (from-radix hms nil (list 24 60 60)) 24))		3	;; Fixed date of Egyptian date e-date.	
			4	(let* ((month (standard-month e-date))	
			5	(day (standard-day e-date))	
1	(defun clock-from-moment (tee)	(1.44)	6	(year (standard-year e-date)))	
2	;; TYPE moment -> clock-time		7	(+ egyptian-epoch ; Days before start of calendar	
3	;; Clock time hour:minute:second from moment tee.		8	(* 365 (1- year)); Days in prior years	
4	(rest (to-radix tee nil (list 24 60 60))))		9	(* 30 (1- month)); Days in prior months this year	
			10	day -1))) ; Days so far this month	
		(1.45)			
1	(delun angle-irom-degrees (alpha)	(1.43)			
2	;; TYPE angle -> list-oi-reals		1	(defun alt-fixed-from-equation (e-date)	(1.48)
3	;; List of degrees-arcminutes-arcseconds from angle alpha		2	·· TYPE equation-date -> fixed-date	(11.10)
4	;; in degrees.		3	·· Fixed date of Egyptian date e-date	
5	(iet* ((ums (co-radix (abs aipha) hii (list 60 60))))		4	(+ emptian-epoch	
0	(ii (>= aipna U)		5	(sigma ((a (list 365 30 1))	
7			6	(e-date e-date))	
8	(11st ; aegrees-minutes-seconds		7	(* a (1 - e - date)))))	
9	(- (IIrst dms)) (- (second dms)) (- (third dms))))))		'	(a (1- e-uale)))))	

(1.52)

	(defun emmtian-from-fixed (date)	(1.49)	6	(vear (standard-vear a-d
2	·· TYPE fixed-date -> envotian-date	(1.4))	7	(+ armenian-epoch
- 3	:: Egyptian equivalent of fixed date.		8	(- (fixed-from-egyptian
4	(let* ((days : Elapsed days since epoch.		9	(egyptian-date year mo
5	(- date equptian-epoch))		10	eqvptian-epoch))))
6	(year ; Year since epoch.			
7	(1+ (quotient days 365)))			
8	(month; Calculate the month by division.		1	(defun armenian-from-fixed (date)
9	(1+ (quotient (mod days 365)		2	TYPE fived-date -> armenian-
10	30)))		3	·· Armenian equivalent of fixed
11	(day ; Calculate the day by subtraction.		4	(equotian-from-fixed
12	(- days		5	(+ date (- equation-epoch arme
13	(* 365 (1- year))		5	() date (egyptian epoen arme
14	(* 30 (1- month))			
15	-1)))			D.1.4 Cycle
16	(egyptian-date year month day)))			Dinit often
			1	(defun kday-on-or-before (k date)
			2	;; TYPE (day-of-week fixed-date
1	(defun armenian-date (year month day)		3	;; Fixed date of the k-day on o
2	;; TYPE (armenian-year armenian-month armenian-day)		4	;; k=0 means Sunday, k=1 means
3	;; TYPE -> armenian-date		5	(- date (day-of-week-from-fixed
4	(list year month day))			
			1	(defun kday-on-or-after (k date)
1	(defconstant armenian-epoch	(1.50)	2	;; TYPE (day-of-week fixed-date
2	;; TYPE fixed-date		3	;; Fixed date of the k-day on o
3	;; Fixed date of start of the Armenian calendar.		4	;; $k=0$ means Sunday, $k=1$ means
4	;; = July 11, 552 CE (Julian).		5	(kday-on-or-before k (+ date 6)
5	(rd 201443))			
			1	(defun kday-nearest (k date)
1	(defun fixed-from-armenian (a-date)	(1.51)	2	;; TYPE (day-of-week fixed-date
2	;; TYPE armenian-date -> fixed-date		3	;; Fixed date of the k-day near
3	;; Fixed date of Armenian date a-date.		4	;; $k=0$ means Sunday, $k=1$ means
4	(let* ((month (standard-month a-date))		5	(kday-on-or-before k (+ date 3)
5	(day (standard-day a-date))			

```
(year (standard-year a-date)))
(+ armenian-epoch
   (- (fixed-from-egyptian
      (egyptian-date year month day))
      egyptian-epoch))))
```

TYPE fixed-date -> armenian-date Armenian equivalent of fixed date. egyptian-from-fixed (+ date (- egyptian-epoch armenian-epoch))))

D.1.4 Cycles of Days

	1	(defun kday-on-or-before (k date)	(1.62)
	2	;; TYPE (day-of-week fixed-date) -> fixed-date	
	3	;; Fixed date of the k-day on or before fixed date.	
	4	;; $k=0$ means Sunday, $k=1$ means Monday, and so on.	
	5	(- date (day-of-week-from-fixed (- date k))))	
	1	(defun kday-on-or-after (k date)	(1.65)
(1.50)	2	;; TYPE (day-of-week fixed-date) -> fixed-date	
	3	;; Fixed date of the k-day on or after fixed date.	
	4	;; $k=0$ means Sunday, $k=1$ means Monday, and so on.	
	5	(kday-on-or-before k (+ date 6)))	
	1	(defun kday-nearest (k date)	(1.66)
(1.51)	2	;; TYPE (day-of-week fixed-date) -> fixed-date	
	3	;; Fixed date of the k-day nearest fixed date.	
	4	;; $k=0$ means Sunday, $k=1$ means Monday, and so on.	
	5	(kday-on-or-before k (+ date 3)))	

1	(defun kday-before (k date)	(1.67)	1	(defun akan-name-difference (a-name1 a-name2)	(1.77)
2	;; TYPE (day-of-week fixed-date) -> fixed-date		2	;; TYPE (akan-name akan-name) -> nonnegative-integer	
3	;; Fixed date of the k-day before fixed date.		3	;; Number of names from Akan name a-name1 to the	
4	;; $k=0$ means Sunday, $k=1$ means Monday, and so on.		4	;; next occurrence of Akan name a-name2.	
5	(kday-on-or-before k (- date 1)))		5	(let* ((prefix1 (akan-prefix a-name1))	
			6	(prefix2 (akan-prefix a-name2))	
			7	(stem1 (akan-stem a-name1))	
1	(defun kdav-after (k date)	(1.68)	8	(stem2 (akan-stem a-name2))	
2	:: TYPE (day-of-week fixed-date) -> fixed-date	(2100)	9	(prefix-difference (- prefix2 prefix1))	
3	;; Fixed date of the k-day after fixed date.		10	(stem-difference (- stem2 stem1)))	
4	:: k=0 means Sunday, k=1 means Monday, and so on.		11	(amod (+ prefix-difference	
5	(kdav-on-or-before k (+ date 7)))		12	(* 36 (- stem-difference	
			13	<pre>prefix-difference)))</pre>	
			14	42)))	
	D.1.5 Akan Calendar				
1	(defun akan-day-name (n)	(1.76)	1	(defconstant akan-day-name-epoch	(1.78)
2	;; TYPE integer -> akan-name		2	;; TYPE fixed-date	
3	;; The <i>n</i> -th name of the Akan cycle.		3	;; RD date of an epoch (day 0) of Akan day cycle.	
4	(akan-name (amod n 6)		4	(rd 37))	
5	(amod n 7)))				
			1	(defun akan-name-from-fixed (date)	(1.79)
1	(defun akan-name (prefix stem)		2	;; TYPE fixed-date -> akan-name	
2	;; TYPE (akan-prefix akan-stem) -> akan-name		3	;; Akan name for date.	
3	(list prefix stem))		4	(akan-day-name (- date akan-day-name-epoch)))	
1	(defun akan-prefix (name)		1	(defun akan-day-name-on-or-before (name date)	(1.80)
2	;; TYPE akan-name -> akan-prefix		2	;; TYPE (akan-name fixed-date) -> fixed-date	
3	(first name))		3	;; Fixed date of latest date on or before fixed date	
			4	;; that has Akan <i>name</i> .	
			5	(mod3	
1	(defun akan-stem (name)		6	(akan-name-difference (akan-name-from-fixed 0) name)	
2	;; TYPE akan-name -> akan-stem		7	date (- date 42)))	
3	(second name))				

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029

	D.2 The Gregorian Calendar		1	(defconstant may	(2.8)
1	(defun gregorian-date (year month day)		2	;; TYPE standard-month	
2	;; TYPE (gregorian-year gregorian-month gregorian-day)		3	;; May on Julian/Gregorian calendar.	
3	;; TYPE -> gregorian-date		4	5)	
4	(list year month day))				
			1	(defconstant june	(2.9)
1	(defconstant gregorian-epoch	(2.3)	2	;; TYPE standard-month	
2	;; TYPE fixed-date		3	;; June on Julian/Gregorian calendar.	
3	;; Fixed date of start of the (proleptic) Gregorian		4	6)	
4	;; calendar.				
5	(rd 1))				
			1	(defconstant july	(2.10)
			2	;; TYPE standard-month	
1	(defconstant january	(2.4)	3	;; July on Julian/Gregorian calendar.	
2	;; TYPE standard-month		4	7)	
3	;; January on Julian/Gregorian calendar.				
4	1)				
			1	(defconstant august	(2.11)
			2	;; TYPE standard-month	
1	(defconstant february	(2.5)	3	;; August on Julian/Gregorian calendar.	
2	;; TYPE standard-month		4	8)	
3	;; February on Julian/Gregorian calendar.				
4	2)				
			1	(defconstant september	(2.12)
		(2.6)	2	;; TYPE standard-month	
1	(derconstant march	(2.0)	3	;; September on Julian/Gregorian calendar.	
2	;; TYPE standard-month		4	9)	
3	;; March on Julian/Gregorian Carendar.				
4	2)				
			1	(defconstant october	(2.13)
1	(defconstant april	(2.7)	2	;; TYPE standard-month	
2	;; TYPE standard-month		3	;; October on Julian/Gregorian calendar.	
3	;; April on Julian/Gregorian calendar.		4	10)	
4	4)				

1 2 3 4 1 2 3	<pre>(defconstant november ;; TYPE standard-month ;; November on Julian/Gregorian calendar. 11) (defconstant december ;; TYPE standard-month :: December on Julian/Gregorian calendar.</pre>	(2.14)	16 17 18 19 20 21 22 23	<pre>(- (* 367 month) 362);assuming 30-day Feb 12) (if (<= month 2) ; Correct for 28- or 29-day Feb 0 (if (gregorian-leap-year? year)</pre>	
4 1 2 3	<pre>12) (defun gregorian-leap-year? (g-year) ;; TYPE gregorian-year -> boolean ;; True if g-year is a leap year on the Gregorian</pre>	(2.16)	1 2 3 4 5	<pre>(defun gregorian-new-year (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of January 1 in g-year. (fixed-from-gregorian (gregorian-date g-year january 1)))</pre>	(2.18)
4 5 6 7	<pre>;; calendar. (and (= (mod g-year 4) 0) (not (member (mod g-year 400)</pre>		1 2 3 4 5	<pre>(defun gregorian-year-end (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of December 31 in g-year. (fixed-from-gregorian (gregorian-date g-year december 31)))</pre>	(2.19)
1 2 3 4 5 6 7 8	<pre>(defun fixed-from-gregorian (g-date) ;; TYPE gregorian-date -> fixed-date ;; Fixed date equivalent to the Gregorian date g-date. (let* ((month (standard-month g-date)) (day (standard-day g-date)) (year (standard-year g-date))) (+ (1- gregorian-epoch); Days before start of calendar (* 365 (1- year)); Ordinary days since epoch</pre>	(2.17)	1 2 3 4 5	<pre>(defun gregorian-year-range (g-year) ;; TYPE gregorian-year -> range ;; The range of moments in Gregorian year g-year. (interval (gregorian-new-year g-year)</pre>	(2.20)
9 10 11 12 13 14 15	<pre>(quotient (1- year)</pre>		1 2 3 4 5 6 7	<pre>(defun gregorian-year-from-fixed (date) ;; TYPE fixed-date -> gregorian-year ;; Gregorian year corresponding to the fixed date. (let* ((d0 ; Prior days.</pre>	(2.21)

8	(d1 ; Prior days not in n400.		15	(quotient	
9	(mod d0 146097))		16	(+ (* 12 (+ prior-days correction)) 373)	
10	(n100 ; 100-year cycles not in n400.		17	367))	
11	(quotient d1 36524))		18	(day ; Calculate the day by subtraction.	
12	(d2 ; Prior days not in n400 or n100.		19	(1+ (- date	
13	(mod d1 36524))		20	(fixed-from-gregorian	
14	<pre>(n4 ; 4-year cycles not in n400 or n100.</pre>		21	(gregorian-date year month 1))))))	
15	(quotient d2 1461))		22	(gregorian-date year month day)))	
16	(d3 ; Prior days not in n400, n100, or n4.				
17	(mod d2 1461))				(2.24)
18	(n1 ; Years not in n400, n100, or n4.		1	(defun gregorian-date-difference (g-datel g-date2)	(2.24)
19	(quotient d3 365))		2	;; TYPE (gregorian-date gregorian-date) -> integer	
20	(year (+ (* 400 n400)		3	;; Number of days from Gregorian date <i>g-datel</i> until	
21	(* 100 n100)		4	;; g-date2.	
22	(* 4 n4)		5	(- (fixed-from-gregorian g-date2)	
23	n1)))		6	(fixed-from-gregorian g-date1)))	
24	(if (or (= n100 4) (= n1 4))				
25	year ; Date is day 366 in a leap year.		1	(defun dav-number (g-date)	(2.25)
26	(1+ year)))); Date is ordinal day (1+ (mod d3 365))		2	·· TYPE gregorian_date -> positive_integer	(2.25)
27	; in (1+ year).		3	Day number in year of Gregorian date <i>g</i> -date	
			4	(grogorian-dato-difforongo	
			5	(gregorian_date (1_ (standard_vear_g_date)) december 31)	
			6	(gregorian-date (i- (standard-year g-date)) december 51)	
1	(defun gregorian-from-fixed (date)	(2.23)	0	g-date;;	
2	;; TYPE fixed-date -> gregorian-date				
3	;; Gregorian (year month day) corresponding to fixed date.		1	(defun days-remaining (g-date)	(2.26)
4	(let* ((year (gregorian-year-from-fixed date))		2	;; TYPE gregorian-date -> nonnegative-integer	
5	(prior-days; This year		3	;; Days remaining in year after Gregorian date g-date.	
6	(- date (gregorian-new-year year)))		4	(gregorian-date-difference	
7	(correction; To simulate a 30-day Feb		5	g-date	
8	(if (< date (fixed-from-gregorian		6	(gregorian-date (standard-year g-date) december 31)))	
9	(gregorian-date year march 1)))				
10	0				
11	(if (gregorian-leap-year? year)		1	(defun last-day-of-gregorian-month (g-year g-month)	(2.27)
12	1		2	;; TYPE (gregorian-year gregorian-month) -> gregorian-day	
13	2)))		3	;; Last day of month g-month in Gregorian year g-year.	
14	(month ; Assuming a 30-day Feb		4	(gregorian-date-difference	

5	(gregorian-date g-year g-month 1)		7	date
6	(gregorian-date (if (= g-month 12)		8	306)))
7	(1+ g-year)		9	(prior-days
8	g-year)		10	(- date (fixed-from-gregorian
9	(amod (1+ g-month) 12)		11	(gregorian-date (1- y) march 1))))
10	1)))		12	(month
			13	(amod (+ (quotient
			14	(+ (* 5 prior-days) 2)
1	(defun alt-fixed-from-gregorian (g-date)	(2.28)	15	153)
2	;; TYPE gregorian-date -> fixed-date		16	3)
3	;; Alternative calculation of fixed date equivalent to the		17	12))
4	;; Gregorian date g-date.		18	(year (- y (quotient (+ month 9) 12)))
5	(let* ((month (standard-month g-date))		19	(day
6	(day (standard-day g-date))		20	(1+ (- date
7	(year (standard-year g-date))		21	(fixed-from-gregorian
8	(m-prime (mod (- month 3) 12))		22	(gregorian-date year month 1))))))
9	(y-prime (- year (quotient m-prime 10))))		23	(gregorian-date year month day)))
10	(+ (1- gregorian-epoch)			
11	-306 ; Days in MarchDecember.		1	(defun alt-gregorian-year-from-fixed (date)
12	(* 365 y-prime); Ordinary days.		2	;; TYPE fixed-date -> gregorian-year
13	(sigma ((y (to-radix y-prime (list 4 25 4)))		3	;; Gregorian year corresponding to the fixed date.
14	(a (list 97 24 1 0)))		4	(let* ((approx ; approximate year
15	(* y a))		5	(quotient (- date gregorian-epoch -2)
16	(quotient ; Days in prior months.		6	146097/400))
17	(+ (* 3 m-prime) 2)		7	(start ; start of next year
18	5)		8	(+ gregorian-epoch
19	(* 30 m-prime)		9	(* 365 approx)
20	day))) ; Days so far this month.		10	(sigma ((y (to-radix approx (list 4 25 4)))
			11	(a (list 97 24 1 0)))
			12	(* y a)))))
1	(defun alt-gregorian-from-fixed (date)	(2.29)	13	(if (< date start)
2	;; TYPE fixed-date -> gregorian-date		14	approx
3	;; Alternative calculation of Gregorian (year month day)		15	(1+ approx)))))
4	;; corresponding to fixed date.			
5	(let* ((y (gregorian-year-from-fixed		1	(defun independence-day (g-year)
6	(+ (1- gregorian-epoch)		2	;; TYPE gregorian-year -> fixed-date

(2.30)

(2.32)

3	;; Fixed date of United States Independence Day in		1	(defun labor-day (g-year)	(2.36)
5	(fixed-from-gregorian (gregorian-date g-year july 4)))		3 4 5	<pre>;; Fixed date of United States Labor Day in Gregorian ;; year g-year (the first Monday in September). (first-kday monday (gregorian-date g-year september 1)))</pre>	
1 2 3 4 5 6 7 8	<pre>(defun nth-kday (n k g-date) ;; TYPE (integer day-of-week gregorian-date) -> fixed-date ;; If n>0, return the n-th k-day on or after ;; g-date. If n<0, return the n-th k-day on or ;; before g-date. If n=0 return bogus. A k-day of ;; 0 means Sunday, 1 means Monday, and so on. (cond ((> n 0)</pre>	(2.33)	1 2 3 4 5	(defun memorial-day (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of United States Memorial Day in Gregorian ;; year g-year (the last Monday in May). (last-kday monday (gregorian-date g-year may 31)))	(2.37)
9 10 11 12 13	<pre>(k(x) f) (kday-before k (fixed-from-gregorian g-date)))) ((< n 0) (+ (* 7 n) (kday-after k (fixed-from-gregorian g-date)))) (t bogus)))</pre>		1 2 3 4 5 6	<pre>(defun election-day (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of United States Election Day in Gregorian ;; year g-year (the Tuesday after the first Monday in ;; November). (first-kday tuesday (gregorian-date g-year november 2)))</pre>	(2.38)
1 2 3 4 5 6	<pre>(defun first-kday (k g-date) ;; TYPE (day-of-week gregorian-date) -> fixed-date ;; Fixed date of first k-day on or after Gregorian date ;; g-date. A k-day of 0 means Sunday, 1 means Monday, ;; and so on. (nth-kday 1 k g-date))</pre>	(2.34)	1 2 3 4 5 6	<pre>(defun daylight-saving-start (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of the start of United States daylight ;; saving time in Gregorian year g-year (the second ;; Sunday in March). (nth-kday 2 sunday (gregorian-date g-year march 1)))</pre>	(2.39)
1 2 3 4 5 6	<pre>(defun last-kday (k g-date) ;; TYPE (day-of-week gregorian-date) -> fixed-date ;; Fixed date of last k-day on or before Gregorian date ;; g-date. A k-day of 0 means Sunday, 1 means Monday, ;; and so on. (nth-kday -1 k g-date))</pre>	(2.35)	1 2 3 4 5 6	<pre>(defun daylight-saving-end (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of the end of United States daylight saving ;; time in Gregorian year g-year (the first Sunday in ;; November). (first-kday sunday (gregorian-date g-year november 1)))</pre>	(2.40)

1	(defun christmas (g-year)	(2.41)	14	(unlucky-fridays-in-range	
2	;; TYPE gregorian-year -> fixed-date		15	(interval (1+ fri) b)))	
3	;; Fixed date of Christmas in Gregorian year g-year.		16	nil)))	
4	(fixed-from-gregorian				
5	(gregorian-date g-year december 25)))		1	(defum umbught fridays (a year)	(2.45)
			2	(defun unidexy-fildays (g-year)	(2.43)
			2	;; IIPE gregorian-year -> IISt-of-IIXed-dates	
1	(defun advent (g-year)	(2.42)	3	;; List of Fridays within Gregorian year g-year	
2	;; TYPE gregorian-year -> fixed-date		4	;; that are day is of Gregorian months.	
3	;; Fixed date of Advent in Gregorian year g-year		5	(unlucky-iridays-in-range	
4	;; (the Sunday closest to November 30).		0	(gregorian-year-range g-year)))	
5	(kday-nearest sunday				
6	(fixed-from-gregorian			D.2. The Islan Colordan	
7	(gregorian-date g-year november 30))))			D.3 The Julian Calendar	
			In the	Lisp code we use $-n$ for year n b.c.e. (Julian):	
			1	(defun bce (n)	
1	(defun epiphany (g-year)	(2.43)	2	;; TYPE standard-year -> julian-year	
2	;; TYPE gregorian-year -> fixed-date		3	;; Negative value to indicate a BCE Julian year.	
3	;; Fixed date of Epiphany in U.S. in Gregorian year		4	(- n))	
4	;; g-year (the first Sunday after January 1).				
5	(first-kday sunday (gregorian-date g-year january 2)))		and po	sitive numbers for c.e. (Julian) years:	
			1	(defun ce (n)	
			2	;; TYPE standard-year -> julian-year	
1	(defun unlucky-fridays-in-range (range)	(2.44)	3	;; Positive value to indicate a CE Julian year.	
2	;; TYPE range -> list-of-fixed-dates		4	n)	
3	;; List of Fridays within <i>range</i> of dates				
4	;; that are day 13 of Gregorian months.		1	(defun julian-date (year month day)	
5	(let* ((a (begin range))		2	:: TYPE (julian-vear julian-month julian-day)	
6	(b (end range))		3	·· TYPE -> julian-date	
7	(fri (kday-on-or-after friday a))		4	(list year month day))	
8	(date (gregorian-from-fixed fri)))			(1100 jeur monen dag),	
9	(if (in-range? fri range)				
10	(append		1	(defun julian-leap-year? (j-year)	(3.1)
11	(if (= (standard-day date) 13)		2	;; TYPE julian-year -> boolean	
12	(list fri)		3	;; True if <i>j-year</i> is a leap year on the Julian calendar.	
13	nil)		4	(= (mod j-year 4) (if (> j-year 0) 0 3)))	

1	(defconstant julian-epoch	(3.2)	8	(1- approx) ; No year 0.	
2	;; TYPE fixed-date		9	approx))	
3	;; Fixed date of start of the Julian calendar.		10	(prior-days; This year	
4	(fixed-from-gregorian (gregorian-date 0 december 30)))		11	(- date (fixed-from-julian	
			12	(julian-date year january 1))))	
			13	(correction; To simulate a 30-day Feb	
1	(defun fixed-from-julian (j-date)	(3.3)	14	(if (< date (fixed-from-julian	
2	:: TYPE julian-date -> fixed-date	(212)	15	(julian-date year march 1)))	
3	:: Fixed date equivalent to the Julian date <i>i</i> -date.		16	0	
4	(let* ((month (standard-month i-date))		17	(if (julian-leap-year? year)	
5	(dav (standard-dav i-date))		18	1	
6	(vear (standard-vear i-date))		19	2)))	
7	(y (if (< vear 0))		20	(month ; Assuming a 30-day Feb	
8	(1+ vear) : No vear zero		21	(quotient	
9	vear)))		22	(+ (* 12 (+ prior-days correction)) 373)	
10	(+ (1- julian-epoch) ; Days before start of calendar		23	367))	
11	(* 365 (1- y)) ; Ordinary days since epoch.		24	(day ; Calculate the day by subtraction.	
12	(quotient (1- y) 4); Leap days since epoch		25	(1+ (- date	
13	(quotient ; Days in prior months this year		26	(fixed-from-julian	
14	(- (* 367 month) 362);assuming 30-day Feb		27	(julian-date year month 1))))))	
15	12)		28	(julian-date year month day)))	
16	(if (<= month 2) ; Correct for 28- or 29-day Feb				
17	0		1	(defconstant kalends	(3.5)
18	(if (julian-leap-year? year)		2	;; TYPE roman-event	
19	-1		3	;; Class of Kalends.	
20	-2))		4	1)	
21	<pre>day))) ; Days so far this month.</pre>				
			1	(defconstant nones	(3.6)
			2	;; TYPE roman-event	
1	(defun julian-from-fixed (date)	(3.4)	3	;; Class of Nones.	
2	;; TYPE fixed-date -> julian-date		4	2)	
3	;; Julian (year month day) corresponding to fixed date.				
4	(let* ((approx ; Nominal year.		1	(defconstant ides	(3.7)
5	(quotient (+ (* 4 (- date julian-epoch)) 1464)		2	;; TYPE roman-event	
6	1461))		3	;; Class of Ides.	
7	(year (if (<= approx 0)		4	3)	

1	(defun roman-date (year month event count leap)		1	(defun nones-of-month (month)	(3.9)
2	;; TYPE (roman-year roman-month roman-event roman-count		2	;; TYPE roman-month -> nones	
3	;; TYPE roman-leap) -> roman-date		3	;; Date of Nones in Roman month.	
4	(list year month event count leap))		4	(- (ides-of-month month) 8))	
1	(defun roman-year (date)		1	(defun fixed-from-roman (r-date)	(3.10)
2	;; TYPE roman-date -> roman-year		2	;; TYPE roman-date -> fixed-date	
3	(first date))		3	;; Fixed date for Roman name r-date.	
			4	(let* ((leap (roman-leap r-date))	
1	(defun roman-month (date)		5	(count (roman-count r-date))	
2	:: TYPE roman-date -> roman-month		6	(event (roman-event r-date))	
3	(second date))		7	(month (roman-month r-date))	
			8	(year (roman-year r-date)))	
			9	(+ (cond	
1	(defun roman-event (date)		10	((= event kalends)	
2	;; TYPE roman-date -> roman-event		11	(fixed-from-julian (julian-date year month 1)))	
3	(third date))		12	((= event nones)	
			13	(fixed-from-julian	
1	(defun roman-count (date)		14	(julian-date year month (nones-of-month month))))	
2	;; TYPE roman-date -> roman-count		15	((= event ides)	
3	(fourth date))		16	(fixed-from-julian	
			17	(julian-date year month (ides-of-month month)))))	
1	(defun roman-leap (date)		18	(- count)	
2	:: TYPE roman-date -> roman-leap		19	(if (and (julian-leap-year? year)	
3	(fifth date))		20	(= month march)	
	(,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,		21	(= event kalends)	
		(2.0)	22	(>= 16 count 6))	
1	(defun ides-of-month (month)	(3.8)	23	0 ; After Ides until leap day	
2	;; TYPE roman-month -> ides		24	1) ; Otherwise	
3	;; Date of Ides in Roman month.		25	(if leap	
4	(if (member month (list march may july october))		26	1 ; Leap day	
5	15		27	0)))) ; Non-leap day	
6	13))				

1	(defun roman-from-fixed (date)	(3.11)	36	(roman-date year march kalends	
2	;; TYPE fixed-date -> roman-date		37	(- 31 day) (= day 25))))))	
3	;; Roman name for fixed date.				
4	(let* ((j-date (julian-from-fixed date))				(2.1.2)
5	(month (standard-month j-date))		1	(defconstant year-rome-founded	(3.12)
6	(day (standard-day j-date))		2	;; TYPE julian-year	
7	(year (standard-year j-date))		3	;; Year on the Julian calendar of the founding of Rome.	
8	(month-prime (amod (1+ month) 12))		4	(bce 753))	
9	(year-prime (if (/= month-prime 1)				
10	year		1	(defun julian-year-from-auc (year)	(3.13)
11	(if (/= year -1)		2	;; TYPE auc-year -> julian-year	
12	(1+ year)		3	;; Julian year equivalent to AUC year	
13	1)))		4	(if (<= 1 year (- year-rome-founded))	
14	(kalends1 (fixed-from-roman		5	(+ year year-rome-founded -1)	
15	(roman-date year-prime month-prime		6	(+ year year-rome-founded)))	
16	kalends 1 false))))				
17	(cond				
18	((= day 1) (roman-date year month kalends 1 false))		1	(defun auc-year-from-julian (year)	(3.14)
19	((<= day (nones-of-month month))		2	;; TYPE julian-year -> auc-year	
20	(roman-date year month nones		3	;; Year AUC equivalent to Julian year	
21	<pre>(1+ (- (nones-of-month month) day)) false))</pre>		4	(if (<= year-rome-founded year -1)	
22	((<= day (ides-of-month month))		5	(- year year-rome-founded -1)	
23	(roman-date year month ides		6	(- year year-rome-founded)))	
24	<pre>(1+ (- (ides-of-month month) day)) false))</pre>				
25	((or (/= month february)		1	(defun olympiad (cycle year)	
26	<pre>(not (julian-leap-year? year)))</pre>		2	·· TYPE (olympiad-cycle olympiad-year) -> olympiad	
27	;; After the Ides, in a month that is not February of a		3	(list cycle year))	
28	;; leap year		5	(libe ejele jeal))	
29	(roman-date year-prime month-prime kalends				
30	<pre>(1+ (- kalends1 date)) false))</pre>		1	(defun olympiad-cycle (o-date)	
31	((< day 25)		2	;; TYPE olympiad -> olympiad-cycle	
32	;; February of a leap year, before leap day		3	(first o-date))	
33	(roman-date year march kalends (- 30 day) false))				
34	(true			(defun elimpiad year (e date)	
35	;; February of a leap year, on or after leap day		1	(actum orympiad-year (o-date)	
			2	;; TIPE OLYMPIAG -> OLYMPIAG-Year	
			3	(secona o-aate))	

1	(defconstant olympiad-start	(3.15)	1	(defconstant autumn	(3.20)
2	;; TYPE julian-year		2	;; TYPE season	
3	;; Start of the Olympiads.		3	;; Longitude of sun at autumnal equinox.	
4	(bce 776))		4	(deg 180))	
1	(defun julian-vear-from-olympiad (o-date)	(3.16)			
2	·· TYPE olympiad -> julian-year	(5.10)	1	(defconstant winter	(3.21)
3	Julian year corresponding to Olympian <i>o-date</i>		2	;; TYPE season	
4	(let* ((cycle (olympiad-cycle o-date))		3	;; Longitude of sun at winter solstice.	
5	(vear (olympiad-year o-date))		4	(deg 270))	
6	(years (+ olympiad-start				
7	(* 4 (1- cvcle))				
8	vear -1)))				(2.22)
9	(if (< years 0)		1	(defun cycle-in-gregorian (season g-year cap-L start)	(3.22)
10	years		2	;; TYPE (season gregorian-year positive-real moment)	
11	(1+ years))))		3	;; TYPE -> list-of-moments	
			4	;; Moments of <i>season</i> in Gregorian year <i>g</i> -year.	
			5	;; Seasonal year is <i>cap-L</i> days, seasons are given as	
1	(defun olympiad-from-julian-year (j-year)	(3.17)	6	;; longitudes and are of equal length,	
2	;; TYPE julian-year -> olympiad		7	;; and a seasonal year started at moment start.	
3	;; Olympiad corresponding to Julian year <i>j</i> -year.		8	(let* ((year (gregorian-year-range g-year))	
4	(let* ((years (- j-year olympiad-start		9	(pos (* (/ season (deg 360)) cap-L))	
5	(if (< j-year 0) 0 1))))		10	(cap-Delta (- pos (mod start cap-L))))	
6	(olympiad (1+ (quotient years 4))		11	(positions-in-range pos cap-L cap-Delta year)))	
7	(1+ (mod years 4))))))				
1	(defconstant spring	(3.18)	1	(defun julian-season-in-gregorian (season g-year)	(3.23)
2	:: TYPE season	(0000)	2	;; TYPE (season gregorian-year) -> list-of-moments	
3	:: Longitude of sun at vernal equinox.		3	;; Moment(s) of Julian season in Gregorian year g-year.	
4	(deg 0))		4	(let* ((cap-Y (+ 365 (hr 6)))	
			5	(offset ; season start	
			6	(* (/ season (deg 360)) cap-Y)))	
1	(defconstant summer	(3.19)	7	(cycle-in-gregorian season g-year cap-Y	
2	;; TYPE season		8	(+ (fixed-from-julian	
3	;; Longitude of sun at summer solstice.		9	(julian-date (bce 1) march 23))	
4	(deg 90))		10	offset))))	

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029

1	(defun julian-in-gregorian (j-month j-day g-year) (3.24) 1	(defconstant coptic-epoch	(4.1)
2	:: TYPE (julian-month julian-day gregorian-year)	2	:: TYPE fixed-date	
3	:: TYPE -> list-of-fixed-dates	3	:: Fixed date of start of the Coptic calendar.	
4	:: List of the fixed dates of Julian month <i>i-month</i> . day	4	(fixed-from-julian (julian-date (ce 284) august 29)))	
5	;; j-day that occur in Gregorian year g-year.		· · · · · · · · · · · · · · · · · · ·	
6	(let* ((jan1 (gregorian-new-year g-year))			
7	(v (standard-vear (julian-from-fixed jan1)))		(defum contin loom very) (a very)	(1 2)
8	(v-prime (if (= v -1)	1	(defui copric=leap-year; (c-year)	(4.2)
9	1	2	;; TIPE copulation boolean	
10	(1 + v)))	3	((mod a succe 4) 2))	
11	:: The possible occurrences in one year are	4	(= (mod c-year 4) 3))	
12	(date0 (fixed-from-julian			
13	(julian-date v j-month j-dav)))			
14	(date1 (fixed-from-julian	1	(defun fixed-from-coptic (c-date)	(4.3)
15	(julian-date v-prime i-month i-day))))	2	;; TYPE coptic-date -> fixed-date	
16	(list-range (list date() date()	3	;; Fixed date of Coptic date <i>c-date</i> .	
17	(areaorian-vear-range a-vear))))	4	(let* ((month (standard-month c-date))	
.,	(gregorian jeur range g jeur),,,	5	(day (standard-day c-date))	
		6	(year (standard-year c-date)))	
		7	(+ coptic-epoch -1 ; Days before start of calendar	
1	(defun eastern-orthodox-christmas (g-year) (3.25) 8	(* 365 (1- year)); Ordinary days in prior years	
2	;; TYPE gregorian-year -> list-of-fixed-dates	9	(quotient year 4); Leap days in prior years	
3	;; List of zero or one fixed dates of Eastern Orthodox	10	(* 30 (1- month)); Days in prior months this year	
4	;; Christmas in Gregorian year g-year.	11	day))) ; Days so far this month	
5	(julian-in-gregorian december 25 g-year))			
		1	(defun coptic-from-fixed (date)	(4.4)
	In languages like Lisp that allow functions as parameters, one could write a generic version of this function	1 2	;; TYPE fixed-date -> coptic-date	
to col	lect the holidays of any given calendar and pass fixed-from-julian to it as an additional parameter	. 3	;; Coptic equivalent of fixed date.	
We ha	ave deliberately avoided this and similar advanced language features in the interests of portability.	4	(let* ((year ; Calculate the year by cycle-of-years formula	
		5	(quotient (+ (* 4 (- date coptic-epoch)) 1463)	
	D.4 The Contic and Ethionic Calendars	6	1461))	
		7	(month; Calculate the month by division.	
1	(defun coput-date (year month day)	8	(1+ (quotient	
2	;; TYPE (coptic-year coptic-month coptic-day) -> coptic-date	9	(- date (fixed-from-coptic	
3	(11st year month day))	10	(coptic-date year 1 1)))	
		11	30)))	

12	(day ; Calculate the day by subtraction.		1	(defun coptic-in-gregorian (c-month c-day g-year)	(4.8)
13	(- date -1		2	;; TYPE (coptic-month coptic-day gregorian-year)	
14	(fixed-from-coptic		3	;; TYPE -> list-of-fixed-dates	
15	(coptic-date year month 1)))))		4	;; List of the fixed dates of Coptic month <i>c-month</i> , day	
16	(coptic-date year month day)))		5	;; c-day that occur in Gregorian year g-year.	
			6	(let* ((jan1 (gregorian-new-year g-year))	
			7	(y (standard-year (coptic-from-fixed jan1)))	
1	(defun ethiopic-date (year month day)		8	;; The possible occurrences in one year are	
2	;; TYPE (ethiopic-year ethiopic-month ethiopic-day)		9	(date0 (fixed-from-coptic	
3	;; TYPE -> ethiopic-date		10	(coptic-date y c-month c-day)))	
4	(list year month day))		11	(date1 (fixed-from-coptic	
			12	(coptic-date (1+ y) c-month c-day))))	
			13	(list-range (list date0 date1)	
1	(defconstant ethiopic-epoch	(4.5)	14	(gregorian-year-range g-year))))	
2	:: TYPE fixed-date	(112)			
3	;; Fixed date of start of the Ethiopic calendar.				
4	(fixed-from-julian (julian-date (ce 8) august 29)))		1	(defun coptic-christmas (g-year)	(4.9)
			2	;; TYPE gregorian-year -> list-of-fixed-dates	
			3	;; List of zero or one fixed dates of Coptic Christmas	
1	(defun fived-from-ethionic (e-date)	(4.6)	4	;; in Gregorian year g-year.	
2	·· TVPE ethionic_date -> fived_date	(4.0)	5	(coptic-in-gregorian 4 29 g-year))	
3	. Fixed date of Ethiopic date e-date				
4	(let* ((month (standard-month e-date))				
5	(day (standard-day e-date))			D.5 The ISO Calendar	
6	(way (standard-year e-date)))		1	(defun iso-date (year week day)	
7	(+ ethiopic-epoch		2	;; TYPE (iso-year iso-week iso-day) -> iso-date	
8	(- (fixed-from-coptic		3	(list year week day))	
9	(coptic-date year month day))				
10	coptic-epoch))))				
			1	(defun iso-week (date)	
			2	;; TYPE iso-date -> iso-week	
1	(defun ethionic-from-fixed (date)	(47)	3	(second date))	
2	·· TYPE fixed-date -> ethionic-date	(4.7)			
3	Ethionic equivalent of fixed date		1	(defun iso-day (date)	
4	(contic-from-fixed		2	TYPE ico-date -> day-of-week	
5	(+ date (- contic-epoch ethionic-epoch))))		3	(third date))	
5			5		

1	(defun iso-year (date)		1	(defun iso-long-year? (i-year)
2	;; TYPE iso-date -> iso-year		2	;; TYPE iso-year -> boolean
3	(first date))		3	;; True if <i>i-year</i> is a long (53-week) year.
			4	(let* ((jan1 (day-of-week-from-fixed
			5	(gregorian-new-year i-year)))
			6	(dec31 (day-of-week-from-fixed
1	(defun fixed-from-iso (i-date)	(5.1)	7	(gregorian-year-end i-year))))
2	;; TYPE iso-date -> fixed-date		8	(or (= jan1 thursday)
3	;; Fixed date equivalent to ISO <i>i-date</i> .		9	(= dec31 thursday))))
4	(let* ((week (iso-week i-date))			(
5	(day (iso-day i-date))			
6	(year (iso-year i-date)))			
7	;; Add fixed date of Sunday preceding date plus day			D.6 The Icelandic Calendar
8	;; in week.		1	(defun icelandic-date (year season week weekday)
9	(+ (nth-kday		2	;; TYPE (icelandic-year icelandic-season
10	week sunday		3	;; TYPE icelandic-week icelandic-weekday) ->
11	(gregorian-date (1- year) december 28)) day)))		4	(list year season week weekday))

- (defun iso-from-fixed (date) (5.2)1 2 ;; TYPE fixed-date -> iso-date 3 ;; ISO (year week day) corresponding to the fixed date. 4 (let* ((approx ; Year may be one too small. (gregorian-year-from-fixed (- date 3))) 5 (year (if (>= date 6 7 (fixed-from-iso 8 (iso-date (1+ approx) 1 1))) 9 (1+ approx) 10 approx)) 11 (week (1+ (quotient 12 (- date 13 (fixed-from-iso (iso-date year 1 1))) 7))) 14 15 (day (amod (- date (rd 0)) 7))) (iso-date year week day))) 16
- 1 (defun icelandic-year (i-date)
 2 ;; TYPE icelandic-date -> icelandic-year
 3 (first i-date))

1 (defun icelandic-season (i-date)

- 2 ;; TYPE icelandic-date -> icelandic-season
- 3 (second i-date))
- 1 (defun icelandic-week (i-date)
- 2 ;; TYPE icelandic-date -> icelandic-week
- 3 (third i-date))
- 1 (defun icelandic-weekday (i-date)
- 2 ;; TYPE icelandic-date -> icelandic-weekday
- 3 (fourth i-date))

(5.3)

icelandic-date

1	(defconstant icelandic-epoch	(6.1)	14	(+ start	
2	;; TYPE fixed-date		15	(* 7 (1- week)) ; Elapsed weeks.	
3	;; Fixed date of start of the Icelandic calendar.		16	(mod (- weekday shift) 7))))	
4	(fixed-from-gregorian (gregorian-date 1 april 19)))				
			1	(defun icelandic-from-fixed (date)	(6.5)
1	(defun icelandic-summer (i-year)	(6.2)	2	;; TYPE fixed-date -> icelandic-date	
2	;; TYPE icelandic-year -> fixed-date		3	;; Icelandic (year season week weekday) corresponding to	
3	;; Fixed date of start of Icelandic year i-year.		4	;; the fixed date.	
4	(let* ((apr19 (+ icelandic-epoch (* 365 (1- i-year))		5	(let* ((approx ; approximate year	
5	(sigma ((y (to-radix i-year (list 4 25 4)))		6	(quotient (- date icelandic-epoch -369)	
6	(a (list 97 24 1 0)))		7	146097/400))	
7	(* y a)))))		8	(year (if (>= date (icelandic-summer approx))	
8	(kday-on-or-after thursday apr19)))		9	approx	
			10	(1- approx)))	
			11	(season (if (< date (icelandic-winter year))	
1	(defun icelandic-winter (i-year)	(6.3)	12	summer	
2	;; TYPE icelandic-year -> fixed-date		13	winter))	
3	;; Fixed date of start of Icelandic winter season		14	(start ; Start of current season.	
4	;; in Icelandic year <i>i-year</i> .		15	(if (= season summer)	
5	(- (icelandic-summer (1+ i-year)) 180))		16	(icelandic-summer year)	
			17	(icelandic-winter year)))	
			18	(week ; Weeks since start of season.	
1	(defun fixed-from-icelandic (i-date)	(6.4)	19	(1+ (quotient (- date start) 7)))	
2	·· TYPE icelandic-date -> fixed-date	(0.1)	20	(weekday (day-of-week-from-fixed date)))	
3	Fixed date equivalent to Icelandic <i>i-date</i>		21	(icelandic-date year season week weekday)))	
4	(let* ((vear (icelandic-vear i-date))				
5	(season (icelandic-season i-date))				
6	(week (icelandic-week i-date))		1	(defun icelandic-leap-year? (i-year)	(6.6)
7	(weekday (icelandic-weekday i-date))		2	;; TYPE icelandic-year -> boolean	
8	(start ; Start of season.		3	;; True if Icelandic <i>i-year</i> is a leap year (53 weeks)	
9	(if (= season summer)		4	;; on the Icelandic calendar.	
10	(icelandic-summer year)		5	(/= (- (icelandic-summer (1+ i-year))	
11	(icelandic-winter year)))		6	(icelandic-summer i-year))	
12	(shift ; First day of week in prior season.		7	364))	
13	(if (= season summer) thursday saturday)))				
1	(defun icelandic-month (i-date)	(6.7)	1	(defun fixed-from-islamic (i-date)	(7.3)
----	--	-------	----	--	-------
2	:: TYPE icelandic-date -> icelandic-month		2	:: TYPE islamic-date -> fixed-date	()
3	:: Month of <i>i-date</i> on the Icelandic calendar.		3	:: Fixed date equivalent to Islamic date <i>i</i> -date.	
4	:: Epagomenae are "month" 0.		4	(let* ((month (standard-month i-date))	
5	(let* ((date (fixed-from-icelandic i-date))		5	(day (standard-day i-date))	
6	(vear (icelandic-year i-date))		6	(year (standard-year i-date)))	
7	(season (icelandic-season i-date))		7	(+ (1- islamic-epoch) ; Days before start of calendar	
8	(midsummer (- (icelandic-winter year) 90))		8	(* (1- year) 354) ; Ordinary days since epoch.	
9	(start (cond ((= season winter)		9	(quotient ; Leap days since epoch.	
10	(icelandic-winter year))		10	(+ 3 (* 11 year)) 30)	
11	((>= date midsummer)		11	(* 29 (1- month)) ; Days in prior months this year	
12	(- midsummer 90))		12	(quotient month 2)	
13	((< date (+ (icelandic-summer year) 90))		13	day))) ; Days so far this month.	
14	(icelandic-summer year))				
15	(t ; Epagomenae.				
16	midsummer))))				
17	(1+ (quotient (- date start) 30))))		1	(defun islamic-from-fixed (date)	(7.4)
			2	;; TYPE fixed-date -> islamic-date	
			3	;; Islamic date (year month day) corresponding to fixed	
	D.7 The Islamic Calendar		4	;; date.	
			5	(let* ((year	
1	(derun islamic-date (year month day)		6	(quotient	
2	;; TYPE (ISIAMIC-year ISIAMIC-MONCH ISIAMIC-day)		7	(+ (* 30 (- date islamic-epoch)) 10646)	
3	(list war worth day)		8	10631))	
4	(list year month day))		9	(prior-days	
			10	(- date (fixed-from-islamic	
1	(defensetant islamic-onoch	(7.1)	11	(islamic-date year 1 1))))	
2	. TYPE fixed-date	(7.1)	12	(month	
2	,, fire fixed date of start of the Islamic calendar		13	(quotient	
4	(fived-from-julian (julian-date (ce 622) july 16)))		14	(+ (* 11 prior-days) 330)	
4	(likeu-liom-julian (julian-date (te 022) july 10)))		15	325))	
			16	(day	
1	(defun islamic-leap-vear? (i-vear)	(7.2)	17	(1+ (- date (fixed-from-islamic	
2	:: TYPE islamic-vear -> boolean	(,)	18	(islamic-date year month 1))))))	
3	:: True if <i>i-year</i> is an Islamic leap year.		19	(islamic-date year month day)))	
4	(< (mod (+ 14 (* 11 i-year)) 30) 11))				

1	(defun islamic-in-gregorian (i-month i-day g-year)	(7.5)	1	(defconstant iyyar	(8.2)
2	;; TYPE (islamic-month islamic-day gregorian-year)		2	;; TYPE hebrew-month	
3	;; TYPE -> list-of-fixed-dates		3	;; Iyyar is month number 2.	
4	;; List of the fixed dates of Islamic month <i>i-month</i> , day		4	2)	
5	;; i-day that occur in Gregorian year g-year.				
6	(let* ((jan1 (gregorian-new-year g-year))				
7	(y (standard-year (islamic-from-fixed jan1)))		1	(defconstant sivan	(8.3)
8	;; The possible occurrences in one year are		2	:: TYPE hebrew-month	(012)
9	(date0 (fixed-from-islamic		3	:: Sivan is month number 3.	
10	(islamic-date y i-month i-day)))		4	3)	
11	(date1 (fixed-from-islamic				
12	(islamic-date (1+ y) i-month i-day)))				
13	(date2 (fixed-from-islamic			(defeendant terming	(8.4)
14	(islamic-date (+ y 2) i-month i-day))))		2	(derconstant tanuaz	(6.4)
15	;; Combine in one list those that occur in current year		3	·· Tammuz is month number 4	
16	(list-range (list date0 date1 date2)		4	() (A)	
17	(gregorian-year-range g-year))))			-,	
		(7.6)	1	(defensiont av	(8.5)
1	(defun mawlid (g-year)	(7.6)	1	(defconstant av	(8.5)
1 2	(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates	(7.6)	1 2 3	(defconstant av ;; TYPE hebrew-month av is month number 5	(8.5)
1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in</pre>	(7.6)	1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5)</pre>	(8.5)
1 2 3 4	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (iclumic in gregorian 2.12 g year))</pre>	(7.6)	1 2 3 4	(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5)	(8.5)
1 2 3 4 5	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year))</pre>	(7.6)	1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5)</pre>	(8.5)
1 2 3 4 5	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year))</pre>	(7.6)	1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul</pre>	(8.5)
1 2 3 4 5	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar</pre>	(7.6)	1 2 3 4 1 2	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month</pre>	(8.5)
1 2 3 4 5	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun bebrew-date (wear month day))</pre>	(7.6)	1 2 3 4 1 2 3	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6.</pre>	(8.5)
1 2 3 4 5	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) TYPE (hebrew-year hebrew-month hebrew-day) => hebrew-date</pre>	(7.6)	1 2 3 4 1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6)</pre>	(8.5)
1 2 3 4 5 1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day))</pre>	(7.6)	1 2 3 4 1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6)</pre>	(8.5)
1 2 3 4 5 1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day))</pre>	(7.6)	1 2 3 4 1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6)</pre>	(8.5)
1 2 3 4 5 1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day))</pre>	(7.6)	1 2 3 4 1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6) (defconstant tishri</pre>	(8.5) (8.6) (8.7)
1 2 3 4 5 1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day)) (defconstant nisan</pre>	(7.6)	1 2 3 4 1 2 3 4 1 2	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6) (defconstant tishri ;; TYPE hebrew-month</pre>	(8.5) (8.6) (8.7)
1 2 3 4 5 1 2 3 1 2	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-gear. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day)) (defconstant nisan ;; TYPE hebrew-month</pre>	(7.6)	1 2 3 4 1 2 3 4 1 2 3	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6) (defconstant tishri ;; TYPE hebrew-month ;; Tishri is month number 7. 7.</pre>	(8.5) (8.6) (8.7)
1 2 3 4 5 1 2 3 1 2 3	<pre>(defun mawlid (g-year) ;; TYPE gregorian-year -> list-of-fixed-dates ;; List of fixed dates of Mawlid an-Nabi occurring in ;; Gregorian year g-year. (islamic-in-gregorian 3 12 g-year)) D.8 The Hebrew Calendar (defun hebrew-date (year month day) ;; TYPE (hebrew-year hebrew-month hebrew-day) -> hebrew-date (list year month day)) (defconstant nisan ;; TYPE hebrew-month ;; Nisan is month number 1.</pre>	(7.6)	1 2 3 4 1 2 3 4 1 2 3 4	<pre>(defconstant av ;; TYPE hebrew-month ;; Av is month number 5. 5) (defconstant elul ;; TYPE hebrew-month ;; Elul is month number 6. 6) (defconstant tishri ;; TYPE hebrew-month ;; Tishri is month number 7. 7)</pre>	(8.5) (8.6) (8.7)

1 2 3 4	(defconstant marheshvan ;; TYPE hebrew-month ;; Marheshvan is month number 8. 8)	(8.8)	1 2 3 4	<pre>(defun hebrew-leap-year? (h-year) ;; TYPE hebrew-year -> boolean ;; True if h-year is a leap year on Hebrew calendar. (< (mod (1+ (* 7 h-year)) 19) 7))</pre>	(8.14)
1 2 3	<pre>(defconstant kislev ;; TYPE hebrew-month ;; Kislev is month number 9.</pre>	(8.9)	1 2 3	<pre>(defun last-month-of-hebrew-year (h-year) ;; TYPE hebrew-year -> hebrew-month ;; Last month of Hebrew year h-gear.</pre>	(8.15)
4	9)		4 5 6	(if (hebrew-leap-year? h-year) adarii adar))	
1 2 3 4	(defconstant tevet ;; TYPE hebrew-month ;; Tevet is month number 10. 10)	(8.10)	1 2	(defun hebrew-sabbatical-year? (h-year) ;; TYPE hebrew-year -> boolean	(8.16)
1	(defconstant shevat	(8.11)	3 4 5	<pre>;; True if h-year is a sabbatical year on the Hebrew ;; calendar. (= (mod h-year 7) 0))</pre>	
2 3 4	;; TYPE hebrew-month ;; Shevat is month number 11. 11)		1 2 3	(defconstant hebrew-epoch ;; TYPE fixed-date ;; Fixed date of start of the Hebrew calendar, that is,	(8.17)
1 2 3	(defconstant adar ;; TYPE hebrew-month ;; Adar is month number 12.	(8.12)	4 5	;; Tishri 1, 1 AM. (fixed-from-julian (julian-date (bce 3761) october 7)))	
4	12)		1 2 3	<pre>(defun molad (h-year h-month) ;; TYPE (hebrew-year hebrew-month) -> rational-moment ;; Moment of mean conjunction of h-month in Hebrew</pre>	(8.19)
1 2 3 4	(defconstant adarii ;; TYPE hebrew-month ;; Adar II is month number 13. 13)	(8.13)	4 5 6 7 8	<pre>;; h-year. (let* ((y ;; Treat Nisan as start of year.</pre>	

9	(months-elapsed		27	;; (quotient (+ 12084 (* months-elapsed 765433))	
10	(+ (- h-month tishri) ;; Months this year.		28	;; 25920)))	
11	(quotient ;; Months until New Year.		29)	
12	(- (* 235 y) 234)		30	(if (< (mod (* 3 (1+ days)) 7) 3); Sun, Wed, or Fri	
13	19))))		31	(+ days 1) ; Delay one day.	
14	(+ hebrew-epoch		32	days)))	
15	-876/25920				
16	(* months-elapsed (+ 29 (hr 12) 793/25920)))))		1	(defun behave year length correction (b year)	(8.21)
			2	(deful hebrew-year-tengen-correction (n-year)	(0.21)
			2	;; IIFE Hebrew-year -> 0-2	
		(8.20)	3	;; belays to start of hebrew year in-year to keep ofdinary	
1	(defun hebrew-calendar-elapsed-days (h-year)	(8.20)	4	;; year in range 555-550 and reap year in range 565-560.	
2	;; TYPE hebrew-year -> integer		5	(net* ((ny0 (nebrew-catendar-etapsed-days (1- n-year)))	
3	;; Number of days elapsed from the (Sunday) noon prior		0	(nyi (nebrew-calendar-elapsed-days n-year))	
4	;; to the epoch of the Hebrew calendar to the mean		/	(nyz (nebrew-calendar-elapsed-days (l+ n-year))))	
5	;; conjunction (molad) of Tisnri of Hebrew year <i>n-year</i> ,		8		
0	;; or one day later.		9	((= (- ny2 ny1) 556) ; Next year would be too long.	
/	(Tet* ((Months-elapsed ; Since start of Hebrew Calendar.		10	(1 - 1)	
8	(quotient (- (* 235 n-year) 234) 19))		11	((= (- nyi nyo) 382) ; Previous year too short.	
9	(parts-elapsed; Fractions of days since prior noon.		12		
10	(+ 12084 (* 13753 months-elapsed)))		13	(t 0))))	
11	(days ; Whole days since prior noon.				
12	(+ (* 29 months-elapsed)		1	(defun hebrew-new-year (h-year)	(8.22)
13	(quotient parts-elapsed 25920)))		2	;; TYPE hebrew-year -> fixed-date	
14	;; If (* 13753 months-elapsed) causes integers that		3	;; Fixed date of Hebrew new year h-year.	
15	;; are too large, use instead:		4	(+ hebrew-epoch	
16	;; (parts-elapsed		5	(hebrew-calendar-elapsed-days h-year)	
17	;; (+ 204 (* 793 (mod months-elapsed 1080))))		6	(hebrew-year-length-correction h-year)))	
18	;; (hours-elapsed				
19	;; (+ 11 (* 12 months-elapsed)				(0.22)
20	;; (* 793 (quotient months-elapsed 1080))		1	(defun last-day-of-hebrew-month (h-year h-month)	(8.23)
21	;; (quotient parts-elapsed 1080)))		2	;; TYPE (hebrew-year hebrew-month) -> hebrew-day	
22	;; (days		3	;; Last day of month h-month in Hebrew year h-year.	
23	;; (+ (* 29 months-elapsed)		4	(if (or (member h-month	
24	;; (quotient hours-elapsed 24)))		5	(list iyyar tammuz elul tevet adarii))	
25	;; It even larger integers aren't a problem, use just:		6	(and (= h-month adar)	
26	;; (days		7	<pre>(not (hebrew-leap-year? h-year)))</pre>	

8	(and (= h-month marheshvan)		10	(< month tishri)	
9	<pre>(not (long-marheshvan? h-year)))</pre>		11	;; Then add days in prior months this year before	
10	(and (= h-month kislev)		12	;; and after Nisan.	
11	(short-kislev? h-year)))		13	(+ (sum (last-day-of-hebrew-month year m)	
12	29		14	m tishri	
13	30))		15	<pre>(<= m (last-month-of-hebrew-year year)))</pre>	
			16	(sum (last-day-of-hebrew-month year m)	
			17	<pre>m nisan (< m month)))</pre>	
1	(defun long-marheshvan? (h-year)	(8.24)	18	;; Else add days in prior months this year	
2	;; TYPE hebrew-year -> boolean		19	(sum (last-day-of-hebrew-month year m)	
3	;; True if Marheshvan is long in Hebrew year h-year.		20	<pre>m tishri (< m month))))))</pre>	
4	(member (days-in-hebrew-year h-year) (list 355 385)))				
			1	(defun hebrew-from-fixed (date)	(8.28)
1	(defun short-kislev? (h-year)	(8.25)	2	;; TYPE fixed-date -> hebrew-date	
2	;; TYPE hebrew-year -> boolean		3	;; Hebrew (year month day) corresponding to fixed date.	
3	;; True if Kislev is short in Hebrew year h-year.		4	;; The fraction can be approximated by 365.25.	
4	(member (days-in-hebrew-year h-year) (list 353 383)))		5	(let* ((approx ; Approximate year	
			6	(1+	
			7	(quotient (- date hebrew-epoch) 35975351/98496)))	
1	(defun days-in-hebrew-year (h-year)	(8.26)	8	;; The value 35975351/98496, the average length of	
2	;; TYPE hebrew-year -> {353,354,355,383,384,385}		9	;; a Hebrew year, can be approximated by 365.25	
3	;; Number of days in Hebrew year h-year.		10	(year ; Search forward.	
4	(- (hebrew-new-year (1+ h-year))		11	(final y (1- approx)	
5	(hebrew-new-year h-year)))		12	<pre>(<= (hebrew-new-year y) date)))</pre>	
			13	(start ; Starting month for search for month.	
			14	(if (< date (fixed-from-hebrew	
1	(defun fixed-from-hebrew (h-date)	(8.27)	15	(hebrew-date year nisan 1)))	
2	;; TYPE hebrew-date -> fixed-date		16	tishri	
3	;; Fixed date of Hebrew date h-date.		17	nisan))	
4	(let* ((month (standard-month h-date))		18	(month ; Search forward from either Tishri or Nisan.	
5	(day (standard-day h-date))		19	(next m start	
6	(year (standard-year h-date)))		20	(<= date	
7	(+ (hebrew-new-year year)		21	(fixed-from-hebrew	
8	day -1 ; Days so far this month.		22	(hebrew-date	
9	(if ;; before Tishri		23	vear	

24	m	5	(let* ((h-year	
25	<pre>(last-day-of-hebrew-month year m))))))</pre>	6	(- g-year	
26	(day ; Calculate the day by subtraction.	7	(gregorian-year-from-fixed hebrew-epoch))))	
27	(1+ (- date (fixed-from-hebrew	8	(fixed-from-hebrew (hebrew-date h-year nisan 15))))	
28	(hebrew-date year month 1))))))			
29	(hebrew-date year month day)))			
must severa	We are using Common Lisp exact arithmetic for rationals here (and elsewhere). Without that facility, or rephrase all quotient operations to work with integers only. The function hebrew-calendar-elapsed-days is called repeatedly during the calculations, ofte al times for the same year. A more efficient algorithm could avoid such repetition. (defun fixed-from-molad (moon) (8.2)	$\begin{array}{c} 1 \\ 2 \\ 3 \\ n \\ 4 \\ 5 \\ 6 \\ 9 \\ 7 \\ 7 \\ 7 \\ 7 \\ 7 \\ 7 \\ 7 \\ 7 \\ 7$	<pre>(defun omer (date) ;; TYPE fixed-date -> omer-count ;; Number of elapsed weeks and days in the omer at date. ;; Returns bogus if that date does not fall during the ;; omer. (let* ((c (- date</pre>	(8.32)
2	;; TIPE duration -> lixed-date	8	(gregorian-year-from-fixed date)))))	
3 4 5 6 7	<pre>;; Fixed date of the moral that occurs moon days ;; and fractional days into the week. (let* ((r (mod (- (* 74377 moon) 2879/2160) 7))) (fixed-from-moment (+ (molad 1 tishri) (* r 765433)))))</pre>	9 10 11	<pre>(if (<= 1 c 49)</pre>	
	(This latter function requires 64-bit integers.)	1 2	(defun purim (g-year) ;; TYPE gregorian-year -> fixed-date	(8.33)
1	(defun yom-kippur (g-year) (8.3	D) ³	;; Fixed date of Purim occurring in Gregorian year g-year.	
2 3 4 5 6 7 8 9	<pre>;; TYPE gregorian-year -> fixed-date ;; Fixed date of Yom Kippur occurring in Gregorian year ;; g-year. (let* ((h-year</pre>	4 5 6 7 8 9	<pre>(let* ((h-year</pre>	
1 2 3	(defun passover (g-year) (8.3 ;; TYPE gregorian-year -> fixed-date ;; Fixed date of Passover occurring in Gregorian year	1) 2 3 4	(defun ta-anit-esther (g-year) ;; TYPE gregorian-year -> fixed-date ;; Fixed date of Ta'anit Esther occurring in ;; Gregorian year g-year.	(8.34)
4	;; g-year.	5	(let* ((purim-date (purim g-year)))	

6	(if ; Purim is on Sunday	12	(list thursday friday))
7	(= (day-of-week-from-fixed purim-date) sunday)	13	;; If Iyyar 4 is Thursday or Friday, then Wednesday
8	;; Then prior Thursday	14	(kday-before wednesday iyyar4))
9	(- purim-date 3)	15	;; If it's on Sunday, then Monday
10	;; Else previous day	16	((= sunday (day-of-week-from-fixed iyyar4))
11	(1- purim-date))))	17	(1+ iyyar4))
		18	(t iyyar4))))

1	(defun tishah-be-av (g-year)	(8.35)			
2	;; TYPE gregorian-year -> fixed-date		1	(defun sh-ela (g-year)	(8.37)
3	;; Fixed date of Tishah be-Av occurring in		2	;; TYPE gregorian-year -> list-of-fixed-dates	
4	;; Gregorian year g-year.		3	;; List of fixed dates of Sh'ela occurring in	
5	(let* ((h-year ; Hebrew year		4	;; Gregorian year g-year.	
6	(- g-year		5	(coptic-in-gregorian 3 26 g-year))	
7	(gregorian-year-from-fixed hebrew-epoch)))				
8	(av9				
9	(fixed-from-hebrew		1	(defun birkath-ha-hama (g-year)	(8.38)
10	(hebrew-date h-year av 9))))		2	;; TYPE gregorian-year -> list-of-fixed-dates	
11	(if ; Ninth of Av is Saturday		3	;; List of fixed date of Birkath ha-Hama occurring in	
12	(= (day-of-week-from-fixed av9) saturday)		4	;; Gregorian year g-year, if it occurs.	
13	;; Then the next day		5	(let* ((dates (coptic-in-gregorian 7 30 g-year)))	
14	(1+ av9)		6	(if (and (not (equal dates nil))	
15	av9)))		7	(= (mod (standard-year	
			8	(coptic-from-fixed (first dates)))	
			9	28)	
1	(defun yom-ha-zikkaron (g-year)	(8.36)	10	17))	
2	;; TYPE gregorian-year -> fixed-date		11	dates	
3	;; Fixed date of Yom ha-Zikkaron occurring in Gregorian		12	nil)))	
4	;; year g-year.				
5	(let* ((h-year ; Hebrew year				
6	(- g-year		1	(defun samuel-season-in-gregorian (season g-year)	(8.39)
7	(gregorian-year-from-fixed hebrew-epoch)))		2	;; TYPE (season gregorian-year) -> list-of-moments	
8	(iyyar4; Ordinarily Iyyar 4		3	;; Moment(s) of season in Gregorian year g-year	
9	(fixed-from-hebrew		4	;; per Samuel.	
10	(hebrew-date h-year iyyar 4))))		5	(let* ((cap-Y (+ 365 (hr 6)))	
11	(cond ((member (day-of-week-from-fixed iyyar4)		6	(offset ; season start	

7	(* (/ season (deg 360)) cap-Y)))		1	(defun ł
8	(cycle-in-gregorian season g-year cap-Y		2	;; TYI
9	(+ (fixed-from-hebrew		3	;; TYI
10	(hebrew-date 1 adar 21))		4	;; Lis
11	(hr 18)		5	;; h-a
12	offset))))		6	(let*
			7	
			8	
1	(defun alt-birkath-ha-hama (g-year)	(8.40)	9	
2	;; TYPE gregorian-year -> list-of-fixed-dates	. ,	10	
3	;; List of fixed date of Birkath ha-Hama occurring in		11	
4	;; Gregorian year g-year, if it occurs.		12	
5	(let* ((cap-Y (+ 365 (hr 6))) ; year		13	
6	(season (+ spring (* (hr 6) (/ (deg 360) cap-Y))))		14	
7	(moments (samuel-season-in-gregorian season g-year)))		15	(lis
8	(if (and (not (equal moments nil))		16	
9	(= (day-of-week-from-fixed (first moments))			
10	wednesday)			
11	(= (time-from-moment (first moments))		1	(defun h
12	(hr 0))) ; midnight		2	;; TYI
13	(list (fixed-from-moment (first moments)))		3	;; Fi>
14	nil)))		4	;; осо
			5	(hebre
1	(defun adda-season-in-gregorian (season g-year)	(8.41)		
2	;; TYPE (season gregorian-year) -> list-of-moments		1	(defun ł
3	;; Moment(s) of <i>season</i> in Gregorian year g-year		2	;; TYI
4	;; per R. Adda bar Ahava.		3	;; Fi>
5	(let* ((cap-Y (+ 365 (hr (+ 5 3791/4104))))		4	;; occ
6	(offset ; season start		5	(let*
7	(* (/ season (deg 360)) cap-Y)))		6	
8	(cycle-in-gregorian season g-year cap-Y		7	
9	(+ (fixed-from-hebrew		8	(if
10	(hebrew-date 1 adar 28))		9	
11	(hr 18)		10	
12	offset))))		11	

(defun hebrew-in	n-gregorian (h-month h-day g-year)	(8.42)
;; TYPE (hebre	ew-month hebrew-day gregorian-year)	
;; TYPE -> 1i	ist-of-fixed-dates	
;; List of the	e fixed dates of Hebrew month <i>h-month</i> , day	
;; h-day that	occur in Gregorian year g-year.	
(let* ((jan1	(gregorian-new-year g-year))	
(y (sta	andard-year (hebrew-from-fixed jan1)))	
;; The	possible occurrences in one year are	
(date0	(fixed-from-hebrew	
	(hebrew-date y h-month h-day)))	
(date1	(fixed-from-hebrew	
	(hebrew-date (1+ y) h-month h-day)))	
(date2	(fixed-from-hebrew	
	(hebrew-date (+ y 2) h-month h-day))))	
(list-range	(list date0 date1 date2)	
	(gregorian-year-range g-year))))	
(defun hanukkah	(g-year)	(8.43)

- 'PE gregorian-year -> list-of-fixed-dates xed date(s) of first day of Hanukkah
- curring in Gregorian year g-year.
- ew-in-gregorian kislev 25 g-year))

1	(defun hebrew-birthday (birthdate h-year)	(8.44)
2	;; TYPE (hebrew-date hebrew-year) -> fixed-date	
3	;; Fixed date of the anniversary of Hebrew birthdate	
4	;; occurring in Hebrew h-year.	
5	(let* ((birth-day (standard-day birthdate))	
6	(birth-month (standard-month birthdate))	
7	(birth-year (standard-year birthdate)))	
8	(if ; It's Adar in a normal Hebrew year or Adar II	
9	; in a Hebrew leap year,	
10	(= birth-month (last-month-of-hebrew-year birth-year))	
11	;; Then use the same day in last month of Hebrew year.	

(8.47)

12	(fixed-from-hebrew		11	;; the day before Kislev 1.
13	(hebrew-date h-year (last-month-of-hebrew-year h-year)		12	((and (= death-month marheshvan)
14	birth-day))		13	(= death-day 30)
15	;; Else use the normal anniversary of the birth date,		14	<pre>(not (long-marheshvan? (1+ death-year))))</pre>
16	;; or the corresponding day in years without that date		15	(1- (fixed-from-hebrew
17	(+ (fixed-from-hebrew		16	(hebrew-date h-year kislev 1))))
18	(hebrew-date h-year birth-month 1))		17	;; If it's Kislev 30 it depends on the first
19	<pre>birth-day -1))))</pre>		18	;; anniversary; if that was not Kislev 30, use
			19	;; the day before Tevet 1.
			20	((and (= death-month kislev)
1	(defun hebrew-birthdav-in-gregorian (birthdate g-vear)	(8.45)	21	(= death-day 30)
2	:: TYPE (hebrew-date gregorian-vear)	()	22	<pre>(short-kislev? (1+ death-year)))</pre>
3	:: TYPE -> list-of-fixed-dates		23	(1- (fixed-from-hebrew
4	;; List of the fixed dates of Hebrew birthday		24	(hebrew-date h-year tevet 1))))
5	;; that occur in Gregorian g-year.		25	;; If it's Adar II, use the same day in last
6	(let* ((jan1 (gregorian-new-year g-year))		26	;; month of Hebrew year (Adar or Adar II).
7	(y (standard-year (hebrew-from-fixed jan1)))		27	((= death-month adarii)
8	;; The possible occurrences in one year are		28	(fixed-from-hebrew
9	(date0 (hebrew-birthday birthdate y))		29	(hebrew-date
10	(date1 (hebrew-birthday birthdate (1+ y)))		30	h-year (last-month-of-hebrew-year h-year)
11	(date2 (hebrew-birthday birthdate (+ y 2))))		31	death-day)))
12	;; Combine in one list those that occur in current year.		32	;; If it's the 30th in Adar I and Hebrew year is not a
13	(list-range (list date0 date1 date2)		33	;; Hebrew leap year (so Adar has only 29 days), use the
14	(gregorian-year-range g-year))))		34	;; last day in Shevat.
			35	((and (= death-day 30)
			36	(= death-month adar)
1	(dofum vabrzoit (doath-dato h-voar)	(8.46)	37	<pre>(not (hebrew-leap-year? h-year)))</pre>
2	(defuil yanizer: (death-date h-year)	(0.40)	38	(fixed-from-hebrew (hebrew-date h-year shevat 30)))
2	, Tire (neblew-date neblew-year) -> Tixed-date		39	;; In all other cases, use the normal anniversary of
4	occurring in Hebrew b-wear		40	;; the date of death.
5	(let+ ((death-day (standard-day death-date))		41	(t (+ (fixed-from-hebrew
6	(death-month (standard-month death-date))		42	(hebrew-date h-year death-month 1))
7	(death-wear (standard-wear death-date)))		43	death-day -1)))))
8	(cond			
9	:: If it's Marheshvan 30 it depends on the first		1	(defun vahrzeit-in-gregorian (death-date g-year)
10	:: anniversary: if that was not Marheshvan 30 use		2	:: TYPE (hebrew-date gregorian-year)
	,,		-	,, (

3	;; TYPE -> list-of-fixed-dates	14	nil)
4	;; List of the fixed dates of death-date (yahrzeit)	15	((and (= h-month kislev) (< h-day 30))
5	;; that occur in Gregorian year g-year.	16	(list monday wednesday friday))
6	(let* ((jan1 (gregorian-new-year g-year))	17	((and (= h-month kislev) (= h-day 30))
7	(y (standard-year (hebrew-from-fixed jan1)))	18	(list monday))
8	;; The possible occurrences in one year are	19	((member h-month (list tevet shevat))
9	(date0 (yahrzeit death-date y))	20	(list sunday monday))
10	(date1 (yahrzeit death-date (1+ y)))	21	((and (= h-month adar) (< h-day 30))
11	(date2 (yahrzeit death-date (+ y 2))))	22	(list sunday monday))
12	;; Combine in one list those that occur in current year	23	(t (list sunday)))))
13	(list-range (list date0 date1 date2)	24	(shift-days (append basic extra) n)))
14	(gregorian-year-range g-year))))		

2 3	;; TYPE (list-of-weekdays integer) -> list-of-weekdays			(
3	. Chift each weekday on ligt 1 by gan Delta days		2	:: TYPE gregorian-year -> fixed-date
	;; SHILL EACH WEEKDAY ON LISE I BY CAP-Della days		3	:: Fixed date of Orthodox Easter in Gregorian year g -uear.
4	(if (equal l nil)		4	(let* ((shifted-epact : Age of moon for April 5.
5	nil		5	(mod (+ 14 (* 11 (mod g-vear 19)))
6	(append (list (mod (+ (first l) cap-Delta) 7))		6	30))
7	(shift-days (rest 1) cap-Delta))))		7	(i-vear (if (> g-vear 0); Julian vear number.
			8	g-year
			9	(1- g-year)))
1	(defun possible-hebrew-days (h-month h-day)	(8.50)	10	(paschal-moon ; Day after full moon on
2	;; TYPE (hebrew-month hebrew-day) -> list-of-weekdays		11	; or after March 21.
3	;; Possible days of week		12	(- (fixed-from-julian (julian-date j-year april 19))
4	(let* ((h-date0 (hebrew-date 5 nisan 1))		13	<pre>shifted-epact)))</pre>
5	;; leap year with full pattern		14	;; Return the Sunday following the Paschal moon.
6	(h-year (if (> h-month elul) 6 5))		15	(kday-after sunday paschal-moon)))
7	(h-date (hebrew-date h-year h-month h-day))			
8	(n (- (fixed-from-hebrew h-date)			
9	(fixed-from-hebrew h-date0)))		1	(defun alt-orthodox-easter (g-year)
10	(basic (list tuesday thursday saturday))		2	;; TYPE gregorian-year -> fixed-date
11	(extra		3	;; Alternative calculation of fixed date of Orthodox Easter
12	(cond		4	;; in Gregorian year g-year.
13	((and (= h-month marheshvan) (= h-day 30))		5	(let* ((paschal-moon ; Day after full moon on

D.9 The Ecclesiastical Calendars

(9.1)

(9.2)

6	; or after March 21.		1	(defun pentecost (g-year)	(9.4)
7	(+ (* 354 g-year)		2	;; TYPE gregorian-year -> fixed-date	
8	(* 30 (quotient (+ (* 7 g-year) 8) 19))		3	;; Fixed date of Pentecost in Gregorian year g-year.	
9	(quotient g-year 4)		4	(+ (easter g-year) 49))	
10	(- (quotient g-year 19))				
11	-273				
12	gregorian-epoch)))			D.10 The Old Hindu Calendars	
13	;; Return the Sunday following the Paschal moon.		1	(defconstant hindu-enoch	(10.1)
14	(kday-after sunday paschal-moon)))		2	·· TYPE fixed-date	(10.1)
			3	;; Fixed date of start of the Hindu calendar (Kali Yuga)	
1	(defun easter (g-vear)	(0.3)	4	(fixed-from-julian (julian-date (bce 3102) february 18)))	
2	TYPE gregorian-year -> fixed-date	(9.5)		· · · · · · · · · · · · · · · · · · ·	
3	Fixed date of Faster in Gregorian year g-year				
4	(let+ ((century (1+ (motient g-vear 100)))		1	(defun hindu-day-count (date)	(10.2)
5	(shifted-enact · Age of moon for April 5		2	;; TYPE fixed-date -> integer	
6	(mod		3	;; Elapsed days (Ahargana) to date since Hindu epoch (KY).	
7	(+ 14 (* 11 (mod g-vear 19)):by Nicaean rule		4	(- date hindu-epoch))	
8	(- :corrected for the Gregorian century rule				
9	(quotient (* 3 century) 4))		1	(defconstant arya-solar-year	(10.3)
10	(quotient;corrected for Metonic		2	;; TYPE rational	
11	; cycle inaccuracy.		3	;; Length of Old Hindu solar year.	
12	(+ 5 (* 8 century)) 25))		4	1577917500/4320000)	
13	30))				
14	(adjusted-epact ; Adjust for 29.5 day month.				(10.4)
15	(if (or (= shifted-epact 0)		1	(derconstant arya-jovian-period	(10.4)
16	(and (= shifted-epact 1)		2	;; HIPE factorial	
17	(< 10 (mod g-year 19))))		4	,, Number of days in one revolution of Supiter around the	
18	(1+ shifted-epact)		5	1577017500/364224)	
19	shifted-epact))		5	15//91/900/904224)	
20	(paschal-moon; Day after full moon on				
21	; or after March 21.		1	(defun jovian-year (date)	(10.5)
22	(- (fixed-from-gregorian		2	;; TYPE fixed-date -> 1-60	
23	(gregorian-date g-year april 19))		3	;; Year of Jupiter cycle at fixed date.	
24	adjusted-epact)))		4	(amod (+ 27 (quotient (hindu-day-count date)	
25	;; Return the Sunday following the Paschal moon.		5	(/ arya-jovian-period 12)))	
26	(kday-after sunday paschal-moon)))		6	60))	

1	(defconstant arya-solar-month	(10.6)	1	(defun old-hindu-lunar-month (date)	
2	;; TYPE rational		2	;; TYPE old-hindu-lunar-date -> old-hindu-lunar-month	
3	;; Length of Old Hindu solar month.		3	(second date))	
4	(/ arya-solar-year 12))				
1	(defun fixed-from-old-hindu-solar (s-date)	(10.7)	1	(defun old-hindu-lunar-lean (date)	
2	;; TYPE hindu-solar-date -> fixed-date		2	TYPE old-hindu-lunar-date -> old-hindu-lunar-leap	
3	;; Fixed date corresponding to Old Hindu solar date s-date.		3	(third date))	
4	(let* ((month (standard-month s-date))		5		
5	(day (standard-day s-date))				
6	(year (standard-year s-date)))				
7	(ceiling		1	(defun old-hindu-lunar-day (date)	
8	(+ hindu-epoch ; Since start of era.		2	;; TYPE old-hindu-lunar-date -> old-hindu-lunar-day	
9	(* year arya-solar-year) ; Days in elapsed years		3	(fourth date))	
10	(* (1- month) arya-solar-month) ;in months.				
11	day (hr -30))))) ; Midnight of day.				
			1	(defun old-hindu-lunar-vear (date)	
1	(defun old-hindu-solar-from-fixed (date)	(10.8)	2	;; TYPE old-hindu-lunar-date -> old-hindu-lunar-year	
2	;; TYPE fixed-date -> hindu-solar-date		3	(first date))	
3	;; Old Hindu solar date equivalent to fixed date.				
4	(let* ((sun ; Sunrise on Hindu date.				
5	(+ (hindu-day-count date) (hr 6)))				
6	(year ; Elapsed years.		1	(defconstant arya-lunar-month	(10.9)
7	(quotient sun arya-solar-year))		2	;; TYPE rational	
8	(month (1+ (mod (quotient sun arya-solar-month)		3	;; Length of Old Hindu lunar month.	
9	12)))		4	1577917500/53433336)	
10	(day (1+ (floor (mod sun arya-solar-month)))))				
11	(hindu-solar-date year month day)))				
			1	(defconstant arya-lunar-day	(10.10)
1	(defun old-hindu-lunar-date (vear month leap day)		2	;; TYPE rational	
2	;; TYPE (old-hindu-lunar-year old-hindu-lunar-month		3	;; Length of Old Hindu lunar day.	
3	;; TYPE old-hindu-lunar-leap old-hindu-lunar-day)		4	(/ arya-lunar-month 30))	
4	;; TYPE -> old-hindu-lunar-date				
5	(list year month leap day))				

1	(defun old-hindu-lunar-leap-year? (l-year)	(10.11)	4	;; 1-date.
2	;; TYPE old-hindu-lunar-year -> boolean		5	(let* ((year (old-hindu-lunar-year l-date))
3	;; True if <i>l-year</i> is a leap year on the		6	(month (old-hindu-lunar-month l-date))
4	;; old Hindu calendar.		7	(leap (old-hindu-lunar-leap l-date))
5	(>= (mod (- (* l-year arya-solar-year)		8	(day (old-hindu-lunar-day l-date))
6	arya-solar-month)		9	(mina ; One solar month before solar new year.
7	arya-lunar-month)		10	(* (1- (* 12 year)) arya-solar-month))
8	23902504679/1282400064))		11	(lunar-new-year ; New moon after mina.
			12	(* arya-lunar-month
			13	<pre>(1+ (quotient mina arya-lunar-month)))))</pre>
1	(defun old-hindu-lunar-from-fixed (date)	(10.13)	14	(ceiling
2	·· TYPE fixed-date -> old-hindu-lunar-date	(10.15)	15	(+ hindu-epoch
3	:: Old Hindu lunar date equivalent to fixed date.		16	lunar-new-year
4	(let* ((sun · Sunrise on Hindu date		17	(* arya-lunar-month
5	(+ (hindu-day-count date) (hr 6)))		18	(if ; If there was a leap month this year.
6	(new-moon : Beginning of lunar month.		19	(and (not leap)
7	(- sun (mod sun arva-lunar-month)))		20	<pre>(<= (ceiling (/ (- lunar-new-year mina)</pre>
8	(leap : If lunar contained in solar.		21	(- arya-solar-month
9	(and (>= (- arva-solar-month arva-lunar-month)		22	arya-lunar-month)))
10	(mod new-moon arva-solar-month))		23	month))
11	(> (mod new-moon arva-solar-month) 0)))		24	month
12	(month : Next solar month's name.		25	(1- month)))
13	(1+ (mod (ceiling (/ new-moon		26	(* (1- day) arya-lunar-day) ; Lunar days.
14	arva-solar-month))		27	(hr -6))))) ; Subtract 1 if phase begins before
15	12)))		28	; sunrise.
16	(day : Lunar days since beginning of lunar month.			
17	(1+ (mod (quotient sun arva-lunar-day) 30)))			
18	(vear : Solar vear at end of lunar month(s).			D.11 The Mayan Calendars
19	(1- (ceiling (/ (+ new-moon arya-solar-month)		1	(defun mayan-long-count-date (baktun katun tun uinal kin)
20	arya-solar-year)))))		2	;; TYPE (mayan-baktun mayan-katun mayan-tun mayan-uinal
21	(old-hindu-lunar-date year month leap day)))		3	;; TYPE mayan-kin) -> mayan-long-count-date
			4	(list baktun katun tun uinal kin))
1	(defun fixed-from-old-hindu-lunar (l-date)	(10.14)	1	(defun mayan-baktun (date)
2	;; TYPE old-hindu-lunar-date -> fixed-date		2	;; TYPE mayan-long-count-date -> mayan-baktun
3	;; Fixed date corresponding to Old Hindu lunar date		3	(first date))

1	(defun mayan-katun (date)		1	(defun mayan-long-count-from-fixed (date)	(11.3)
2	;; TYPE mayan-long-count-date -> mayan-katun		2	;; TYPE fixed-date -> mayan-long-count-date	
3	(second date))		3	;; Mayan long count date of fixed date.	
			4	(to-radix (- date mayan-epoch) (list 20 20 18 20)))	
1	(defun mayan-tun (date)				
2	;; TYPE mayan-long-count-date -> mayan-tun		1	(defun mayan-haab-date (month day)	
3	(third date))		2	;; TYPE (mayan-haab-month mayan-haab-day) -> mayan-haab-date	
			3	(list month day))	
1	(defun mayan-uinal (date)				
2	;; TYPE mayan-long-count-date -> mayan-uinal		1	(defun mayan-haab-day (date)	
3	(fourth date))		2	;; TYPE mayan-haab-date -> mayan-haab-day	
			3	(second date))	
1	(defun mayan-kin (date)				
2	;; TYPE mayan-long-count-date -> mayan-kin		1	(defun mayan-haab-month (date)	
3	(fifth date))		2	;; TYPE mayan-haab-date -> mayan-haab-month	
			3	(first date))	
1	(defconstant mayan-epoch	(11.1)			
2	;; TYPE fixed-date		1	(defun mayan-haab-ordinal (h-date)	(11.4)
3	;; Fixed date of start of the Mayan calendar, according		2	;; TYPE mayan-haab-date -> nonnegative-integer	
4	;; to the Goodman-Martinez-Thompson correlation.		3	;; Number of days into cycle of Mayan haab date h-date.	
5	;; That is, August 11, -3113.		4	(let* ((day (mayan-haab-day h-date))	
6	(fixed-from-jd 584283))		5	(month (mayan-haab-month h-date)))	
			6	(+ (* (1- month) 20) day)))	
1	(defun fixed-from-mayan-long-count (count)	(11.2)			
2	;; TYPE mayan-long-count-date -> fixed-date		1	(defconstant mayan-haab-epoch	(11.5)
3	;; Fixed date corresponding to the Mayan long count,		2	;; TYPE fixed-date	
4	;; which is a list (baktun katun tun uinal kin).		3	;; Fixed date of start of haab cycle.	
5	(+ mayan-epoch ; Fixed date at Mayan 0.0.0.0.0		4	(- mayan-epoch	
6	(from-radix count (list 20 20 18 20))))		5	(mayan-haab-ordinal (mayan-haab-date 18 8))))	

1	(defun mayan-haab-from-fixed (date)	(11.6)	1	(defun mayan-tzolkin-from-fixed (date)	(11.9)
2	;; TYPE fixed-date -> mayan-haab-date		2	;; TYPE fixed-date -> mayan-tzolkin-date	
3	;; Mayan haab date of fixed date.		3	;; Mayan tzolkin date of fixed date.	
4	(let* ((count		4	(let* ((count (- date mayan-tzolkin-epoch -1))	
5	(mod (- date mayan-haab-epoch) 365))		5	(number (amod count 13))	
6	(day (mod count 20))		6	(name (amod count 20)))	
7	(month (1+ (quotient count 20))))		7	(mayan-tzolkin-date number name)))	
8	(mayan-haab-date month day)))				
			1	(defun mayan-tzolkin-ordinal (t-date)	(11.10)
1	(defun mayan-haab-on-or-before (haab date)	(11.7)	2	;; TYPE mayan-tzolkin-date -> nonnegative-integer	
2	;; TYPE (mayan-haab-date fixed-date) -> fixed-date		3	;; Number of days into Mayan tzolkin cycle of <i>t-date</i> .	
3	;; Fixed date of latest date on or before fixed date		4	(let* ((number (mayan-tzolkin-number t-date))	
4	;; that is Mayan haab date haab.		5	(name (mayan-tzolkin-name t-date)))	
5	(mod3 (+ (mayan-haab-ordinal haab) mayan-haab-epoch)		6	(mod (+ number -1	
6	date (- date 365)))		7	(* 39 (- number name)))	
			8	260)))	
1	(defun mayan-tzolkin-date (number name)				
2	;; TYPE (mayan-tzolkin-number mayan-tzolkin-name)		1	(defun mayan-tzolkin-on-or-before (tzolkin date)	(11.11)
3	;; TYPE -> mayan-tzolkin-date		2	;; TYPE (mayan-tzolkin-date fixed-date) -> fixed-date	
4	(list number name))		3	;; Fixed date of latest date on or before fixed date	
			4	;; that is Mayan tzolkin date tzolkin.	
1	(defun maran trallin number (data)		5	(mod3 (+ (mayan-tzolkin-ordinal tzolkin) mayan-tzolkin-epoch)	1
1	(derum mayan-tzoikin-number (date)		6	date (- date 260)))	
2	(first date))				
			1	(defun mayan-year-bearer-from-fixed (date)	(11.12)
1	(defun mayan-tzolkin-name (date)		2	;; TYPE fixed-date -> mayan-tzolkin-name	
2	. TYPE mayan-tzolkin-dato -> mayan-tzolkin-namo		3	;; Year bearer of year containing fixed date.	
2	(cocord date))		4	;; Returns bogus for uayeb.	
5	(second date))		5	(let* ((x (mayan-haab-on-or-before	
			6	(mayan-haab-date 1 0)	
1	(defconstant mayan-tzolkin-epoch	(11.8)	7	date)))	
2	;; TYPE fixed-date		8	(if (= (mayan-haab-month (mayan-haab-from-fixed date))	
3	;; Start of tzolkin date cycle.		9	19)	
4	(- mayan-epoch		10	bogus	
5	(mayan-tzolkin-ordinal (mayan-tzolkin-date 4 20))))		11	(mayan-tzolkin-name (mayan-tzolkin-from-fixed x)))))	

1	(defun mayan-calendar-round-on-or-before (haab tzolkin date)	(11.13)	1	(defun aztec-xihuitl-ordinal (x-date)	(11.15)
2	;; TYPE (mayan-haab-date mayan-tzolkin-date fixed-date)		2	;; TYPE aztec-xihuitl-date -> nonnegative-integer	
3	;; TYPE -> fixed-date		3	;; Number of elapsed days into cycle of Aztec xihuitl x-date.	
4	;; Fixed date of latest date on or before date, that is		4	(let* ((day (aztec-xihuitl-day x-date))	
5	;; Mayan haab date haab and tzolkin date tzolkin.		5	(month (aztec-xihuitl-month x-date)))	
6	;; Returns bogus for impossible combinations.		6	(+ (* (1- month) 20) (1- day))))	
7	(let* ((haab-count				
8	(+ (mayan-haab-ordinal haab) mayan-haab-epoch))				
9	(tzolkin-count				(11.16)
10	(+ (mayan-tzolkin-ordinal tzolkin)		1	(derconstant aztec-xinuiti-correlation	(11.10)
11	mayan-tzolkin-epoch))		2	;; TIPE lixed-date	
12	(diff (- tzolkin-count haab-count)))		3	;; Start of a Xinuiti cycle.	
13	(if (= (mod diff 5) 0)		4	(- aztec-correlation	
14	(mod3 (+ haab-count (* 365 diff))		5	(aztec-xinuiti-ordinal (aztec-xinuiti-date II 2))))	
15	date (- date 18980))				
16	bogus))); haab-tzolkin combination is impossible.				
			1	(defun aztec-xihuitl-from-fixed (date)	(11.17)
1	(defconstant aztec-correlation	(11.14)	2	;; TYPE fixed-date -> aztec-xihuitl-date	
2	;; TYPE fixed-date		3	;; Aztec xihuitl date of fixed date.	
3	;; Known date of Aztec cycles (Caso's correlation)		4	(let* ((count (mod (- date aztec-xihuitl-correlation) 365))	
4	(fixed-from-julian (julian-date 1521 August 13)))		5	(day (1+ (mod count 20)))	
			6	(month (1+ (quotient count 20))))	
1	(defun aztec-xihuitl-date (month day)		7	(aztec-xihuitl-date month day)))	
2	;; TYPE (aztec-xihuitl-month aztec-xihuitl-day) ->				
3	;; TYPE aztec-xihuitl-date				
4	(list month day))				(11.10)
			1	(defun aztec-xinuiti-on-or-before (xinuiti date)	(11.18)
1	(defun aztec-xihuitl-month (date)		2	;; TYPE (aztec-xinuiti-date fixed-date) -> fixed-date	
2	;; TYPE aztec-xihuitl-date -> aztec-xihuitl-month		3	;; Fixed date of latest date on or before fixed date	
3	(first date))		4	;; that is Aztec xihuitl date xihuitl.	
1	(defun aztec-vihuitl-dav (date)		5	(mod3 (+ aztec-xihuitl-correlation	
2	(derum aztec-ximurti-day (date)		6	(aztec-xihuit1-ordinal xihuit1))	
2	(accord date))		7	date (- date 365)))	
2	(Second date/)				

1	(defun aztec-tonalpohualli-date (number name)		3	;; Aztec tonalpohualli date of fixed date.	
2	;; TYPE (aztec-tonalpohualli-number aztec-tonalpohualli-name)		4	(let* ((count (- date aztec-tonalpohualli-correlation -1))	
3	;; TYPE -> aztec-tonalpohualli-date		5	(number (amod count 13))	
4	(list number name))		6	(name (amod count 20)))	
			7	(aztec-tonalpohualli-date number name)))	
1	(defun aztec-tonalpohualli-number (date)				
2	;; TYPE aztec-tonalpohualli-date -> aztec-tonalpohualli-numbe	r	1	(defun aztec-tonalpohualli-on-or-before (tonalpohualli date) (11	.22)
3	(first date))		2	;; TYPE (aztec-tonalpohualli-date fixed-date) -> fixed-date	
			3	;; Fixed date of latest date on or before fixed date	
			4	;; that is Aztec tonalpohualli date tonalpohualli.	
1	(defun aztec-tonalpohualli-name (date)		5	(mod3 (+ aztec-tonalpohualli-correlation	
2	;; TYPE aztec-tonalpohualli-date -> aztec-tonalpohualli-name		6	(aztec-tonalpohualli-ordinal tonalpohualli))	
3	(second date))		7	date (- date 260)))	
1	(defun artec-tonalnohualli-ordinal (t-date)	(11.19)	1	(defun aztec-xiuhmolpilli-designation (number name)	
2	(defun aztec-tonalpohualli-dato -> nonnogativo-integer	(11.19)	2	:: TYPE (aztec-xiuhmolpilli-number aztec-xiuhmolpilli-name)	
2	,, TIPE azcec-conarponualli-date -> hommegative-integer		3	:: TYPE -> aztec-xiubmolpilli-designation	
4	(let+ ((number (aztec=tonalpohualli=number t=date))		4	(list number name))	
5	(name (aztec-tonalpohualli-name t-date)))			· · · · · · · · · · · · · · · · · · ·	
6	(mod (+ number -1				
7	(* 39 (- number name)))		1	(defun aztec-xiuhmolpilli-number (date)	
8	260)))		2	:: TYPE aztec-xiuhmolpilli-designation -> aztec-xiuhmolpilli-numb	ber
			3	(first date))	
1	(defconstant aztec-tonalpohualli-correlation	(11.20)			
2	;; TYPE fixed-date		1	(defun aztec-xiuhmolpilli-name (date)	
3	;; Start of a tonalpohualli date cycle.		2	;; TYPE aztec-xiuhmolpilli-designation -> aztec-xiuhmolpilli-name	2
4	(- aztec-correlation		3	(second date))	
5	(aztec-tonalpohualli-ordinal				
6	(aztec-tonalpohualli-date 1 5))))				
			1	(defun aztec-xiuhmolpilli-from-fixed (date) (11	.23)
			2	;; TYPE fixed-date -> aztec-xiuhmolpilli-designation	
1	(defun aztec-tonalpohualli-from-fixed (date)	(11.21)	3	;; Designation of year containing fixed date.	
2	;; TYPE fixed-date -> aztec-tonalpohualli-date		4	;; Returns bogus for nemontemi.	

5	(let* ((x (aztec-xihuitl-on-or-before	1	(defun bali-luang (b-date)
6	(aztec-xihuitl-date 18 20)	2	;; TYPE balinese-date -> boolean
7	(+ date 364)))	3	(first b-date))
8	(month (aztec-xihuitl-month		
9	<pre>(aztec-xihuitl-from-fixed date))))</pre>		
10	(if (= month 19)	1	(defun bali-dwiwara (b-date)
11	bogus	2	·· TYPE balinese-date -> 1-2
12	(aztec-tonalpohualli-from-fixed x))))	3	(second h-date))
		5	(Second D-date))
1	(defun aztec-xihuitl-tonalpohualli-on-or-before (11	.24)	
2	(xihuitl tonalpohualli date)		
3	;; TYPE (aztec-xihuitl-date aztec-tonalpohualli-date	1	(delun ball-triwara (b-date)
4	;; TYPE fixed-date) -> fixed-date	2	;; TYPE ballnese-date -> 1-3
5	;; Fixed date of latest xihuitl-tonalpohualli combination	3	(third b-date))
6	;; on or before date. That is the date on or before		
7	;; date that is Aztec xihuitl date xihuitl and		
8	;; tonalpohualli date tonalpohualli.	1	(defun bali-caturwara (b-date)
9	;; Returns bogus for impossible combinations.	2	;; TYPE balinese-date -> 1-4
10	(let* ((xihuitl-count	3	(fourth b-date))
11	(+ (aztec-xihuitl-ordinal xihuitl)		
12	aztec-xihuitl-correlation))		
13	(tonalpohualli-count	1	(defun bali-pancawara (b-date)
14	(+ (aztec-tonalpohualli-ordinal tonalpohualli)	2	:: TYPE balinese-date -> 1-5
15	aztec-tonalpohualli-correlation))	3	(fifth b-date))
16	<pre>(diff (- tonalpohualli-count xihuitl-count)))</pre>		
17	(if (= (mod diff 5) 0)		
18	(mod3 (+ xihuitl-count (* 365 diff))		
19	date (- date 18980))	1	(defun bali-sadwara (b-date)
20	bogus))); xihuitl-tonalpohualli combination is impossible.	2	;; TYPE balinese-date -> 1-6
		3	(sixth b-date))

D.12 The Balinese Pawukon Calendar

1	(defun balinese-date (b1 b2 b3 b4 b5 b6 b7 b8 b9 b0)
2	;; TYPE (boolean 1-2 1-3 1-4 1-5 1-6 1-7 1-8 1-9 0-9)
3	;; TYPE -> balinese-date
4	(list b1 b2 b3 b4 b5 b6 b7 b8 b9 b0))

1	(defun bali-saptawara (b-date)
2	;; TYPE balinese-date -> 1-7
3	(seventh b-date))

1	(defun bali-asatawara (b-date)		1	(defun bali-day-from-fixed (date)	(12.3)
2	;; TYPE balinese-date -> 1-8		2	;; TYPE fixed-date -> 0-209	
3	(eighth b-date))		3	;; Position of date in 210-day Pawukon cycle.	
			4	(mod (- date bali-epoch) 210))	
1	(defun bali-sangawara (b-date)				
2	;; TYPE balinese-date -> 1-9		1	(defun bali-triwara-from-fixed (date)	(12.4)
3	(ninth b-date))		2	:: TYPE fixed-date -> 1-3	()
			3	:: Position of <i>date</i> in 3-day Balinese cycle.	
			4	(1+ (mod (bali-day-from-fixed date) 3)))	
1	(defun bali-dasawara (b-date)			· · · • • • · · · ·	
2	;; TYPE balinese-date -> 0-9				
3	(tenth b-date))		1	(dofum bali-sadwara-from-fixed (date)	(12.5)
			2	·· TYPE fixed-date -> 1-6	(12.5)
			3	:: Position of <i>date</i> in 6-day Balinese cycle.	
1	(defun bali-pawukon-from-fixed (date)	(12.1)	4	(1+ (mod (bali-dav-from-fixed date) 6)))	
2	;; TYPE fixed-date -> balinese-date			e e se esta se a se	
3	;; Positions of date in ten cycles of Balinese Pawukon				
4	;; calendar.			(defun hali contavana from fixed (data)	(126)
5	(balinese-date (bali-luang-from-fixed date)		1	(defun ball-saptawara-from-fixed (date)	(12.0)
6	(bali-dwiwara-from-fixed date)		2	;; IIFE IIXed-date -> I-7	
7	(bali-triwara-from-fixed date)		3	;; POSICION OF date in Bailnese week.	
8	(bali-caturwara-from-fixed date)		4	(IT (mou (Dall-day-from-fixed date) /)))	
9	(bali-pancawara-from-fixed date)				
10	(bali-sadwara-from-fixed date)				
11	(bali-saptawara-from-fixed date)		1	(defun bali-pancawara-from-fixed (date)	(12.7)
12	(bali-asatawara-from-fixed date)		2	;; TYPE fixed-date -> 1-5	
13	(bali-sangawara-from-fixed date)		3	;; Position of <i>date</i> in 5-day Balinese cycle.	
14	<pre>(bali-dasawara-from-fixed date)))</pre>		4	(amod (+ (bali-day-from-fixed date) 2) 5))	
1	(defconstant bali-epoch	(12.2)	1	(defun bali-week-from-fixed (date)	(12.8)
2	;; TYPE fixed-date	()	2	;; TYPE fixed-date -> 1-30	
3	;; Fixed date of start of a Balinese Pawukon cycle.		3	;; Week number of date in Balinese cycle.	
4	(fixed-from-jd 146))		4	(1+ (quotient (bali-day-from-fixed date) 7)))	

1	(defun bali-dasawara-from-fixed (date)	(12.9)	5	(1+ (mod	
2	;; TYPE fixed-date -> 0-9		6	(max 6	
3	;; Position of date in 10-day Balinese cycle.		7	(+ 4 (mod (- day 70)	
4	(let* ((i ; Position in 5-day cycle.		8	210)))	
5	<pre>(1- (bali-pancawara-from-fixed date)))</pre>		9	8))))	
6	(j ; Weekday.				
7	<pre>(1- (bali-saptawara-from-fixed date))))</pre>		1	(defun bali-caturwara-from-fixed (date)	(12.14)
8	(mod (+ 1 (nth i (list 5 9 7 4 8))		2	:: TYPE fixed-date -> 1-4	
9	(nth j (list 5 4 3 7 8 6 9)))		3	:: Position of <i>date</i> in 4-day Balinese cycle.	
10	10)))		4	(amod (bali-asatawara-from-fixed date) 4))	
		(12.10)	1	(defun bali-on-or-before (b-date date)	(12.15)
1	(defun bali-dwiwara-from-fixed (date)	(12.10)	2	;; TYPE (balinese-date fixed-date) -> fixed-date	
2	;; TYPE fixed-date -> 1-2		3	;; Last fixed date on or before date with Pawukon b-date.	
3	;; Position of <i>date</i> in 2-day Balinese cycle.		4	(let* ((luang (bali-luang b-date))	
4	(amod (ball-dasawara-from-fixed date) 2))		5	(dwiwara (bali-dwiwara b-date))	
			6	(triwara (bali-triwara b-date))	
			7	(caturwara (bali-caturwara b-date))	
1	(defun bali-luang-from-fixed (date)	(12.11)	8	(pancawara (bali-pancawara b-date))	
2	;; TYPE fixed-date -> boolean		9	(sadwara (bali-sadwara b-date))	
3	;; Membership of <i>date</i> in "1-day" Balinese cycle.		10	(saptawara (bali-saptawara b-date))	
4	(evenp (bali-dasawara-from-fixed date)))		11	(asatawara (bali-asatawara b-date))	
			12	(sangawara (bali-sangawara b-date))	
			13	(dasawara (bali-dasawara b-date))	
1	(defun bali-sangawara-from-fixed (date)	(12.12)	14	(a5 ; Position in 5-day subcycle.	
2	;; TYPE fixed-date -> 1-9		15	(1- pancawara))	
3	;; Position of date in 9-day Balinese cycle.		16	(a6 ; Position in 6-day subcycle.	
4	(1+ (mod (max 0		17	(1- sadwara))	
5	(- (bali-day-from-fixed date) 3))		18	(b7 ; Position in 7-day subcycle.	
6	9)))		19	(1- saptawara))	
			20	(b35 ; Position in 35-day subcycle.	
			21	(mod (+ a5 14 (* 15 (- b7 a5))) 35))	
1	(defun bali-asatawara-from-fixed (date)	(12.13)	22	(days ; Position in full cycle.	
2	;; TYPE fixed-date -> 1-8		23	(+ a6 (* 36 (- b35 a6))))	
3	;; Position of date in 8-day Balinese cycle.		24	(cap-Delta (bali-day-from-fixed (rd 0))))	
4	(let* ((day (bali-day-from-fixed date)))		25	(- date (mod (- (+ date cap-Delta) days) 210))))	

(14.7)

1	(defun kajeng-keliwon (g-year)	(12.16)	1	(defun sin-degrees (theta)
2	;; TYPE gregorian-year -> list-of-fixed-dates		2	;; TYPE angle -> amplitude
3	;; Occurrences of Kajeng Keliwon (9th day of each		3	;; Sine of theta (given in degrees).
4	;; 15-day subcycle of Pawukon) in Gregorian year g-year.		4	(sin (radians-from-degrees theta)))
5	(let* ((year (gregorian-year-range g-year))			
6	<pre>(cap-Delta (bali-day-from-fixed (rd 0))))</pre>		1	(defun cos-degrees (theta)
7	(positions-in-range 8 15 cap-Delta year)))		2	·· TYPE angle => amplitude
			3	Cosine of theta (given in degrees)
			4	(cos (radians-from-degrees theta)))
1	(defun tumpek (g-year)	(12.17)	-	(cos (radians from degrees checa)))
2	;; TYPE gregorian-year -> list-of-fixed-dates			
3	;; Occurrences of Tumpek (14th day of Pawukon and every		1	(defun tan-degrees (theta)
4	;; 35th subsequent day) within Gregorian year g-year.		2	;; TYPE angle -> real
5	(let* ((year (gregorian-year-range g-year))		3	;; Tangent of theta (given in degrees).
6	(cap-Delta (bali-day-from-fixed (rd 0))))		4	(tan (radians-from-degrees theta)))
7	(positions-in-range 13 35 cap-Delta year)))			
			1	(defun arctan-degrees (y x)
	D 13 Conoral Cyclical Calendars		2	;; TYPE (real real) -> angle
	D.15 General Cyclical Calendars		3	: Arctangent of u/x in degrees.
No Lis	sp code is included for this chapter.		4	;; Returns bogus if x and y are both 0.
			5	(if (and (= x y 0))
	D.14 Time and Astronomy		6	bogus
Comm	ion Lisn's built-in trigonometric functions work with radians, whereas we have used degrees. The fi	ollowing	7	(mod
functio	and the necessary normalization and conversions:	Showing	8	(if (= x 0)
runetit	no do de necessary normalization and conversions.		9	(* (sign y) (deg 90L0))
1	(defun radians-from-degrees (theta)		10	(let* ((alpha (degrees-from-radians
2	·· TYPE real -> radian		11	(atan (/ y x)))))
3	·· Convert angle theta from degrees to radians		12	(if (>= x 0)
4	$(+ \pmod{2} + \binom{1}{2} + \binom{1}$		13	alpha
-	(* (mod elleca 500) pr 1/100//		14	(+ alpha (deg 180L0)))))
			15	360)))
1	(defun degrees-from-radians (theta)			
2	·· TYDE radian => angle			
3	Convert angle theta from radians to degrees		1	(defun arcsin-degrees (x)
4	(mod (/ thota pi 1/190) 360))		2	;; TIPE amplitude -> angle
4	(mod (/ checa pr 1/100) 200))		3	;; Arcsine of x in degrees.
			4	(degrees-from-radians (asin x)))

```
1
    (defun arccos-degrees (x)
                                                                                   1
                                                                                        (defun mins (x)
2
      ;; TYPE amplitude -> angle
                                                                                   2
                                                                                          ;; TYPE real -> angle
3
      ;; Arccosine of x in degrees.
                                                                                          ;; x arcminutes
                                                                                   3
       (degrees-from-radians (acos x)))
4
                                                                                   4
                                                                                          (/ x 60))
   We also use the following functions to indicate units; they are also used for typesetting:
                                                                                        (defun secs (x)
                                                                                   1
    (defun hr (x)
1
                                                                                          ;; TYPE real -> angle
                                                                                   2
      ;; TYPE real -> duration
2
                                                                                          ;; x arcseconds
                                                                                   3
      ;; x hours.
3
                                                                                          (/ x 3600))
                                                                                   4
4
      (/ x 24))
    (defun mn (x)
1
                                                                                        (defun angle (d m s)
                                                                                   1
2
      ;; TYPE real -> duration
                                                                                   2
                                                                                          ;; TYPE (integer integer real) -> angle
3
      ;; x minutes.
                                                                                          ;; d degrees, m arcminutes, s arcseconds.
                                                                                   3
4
      (/ x 24 60))
                                                                                          (+ d (/ (+ m (/ s 60)) 60)))
                                                                                   4
1
    (defun sec (x)
2
      ;; TYPE real -> duration
                                                                                   1
                                                                                        (defun degrees-minutes-seconds (d m s)
3
      :: x seconds.
                                                                                   2
                                                                                          ;; TYPE (degree minute real) -> angle
      (/ x 24 60 60))
4
                                                                                          (list d m s))
                                                                                   3
                                                                                  The deg function is also applied to lists, to indicate that it is a list of angles.
    (defun mt (x)
1
                                                                                      The following allow us to specify locations and directions:
      ;; TYPE real -> distance
2
3
      ;; x meters.
4
      ;; For typesetting purposes.
                                                                                   1
                                                                                        (defun location (latitude longitude elevation zone)
                                                                                   2
                                                                                          ;; TYPE (half-circle circle distance real) -> location
5
      x)
                                                                                          (list latitude longitude elevation zone))
                                                                                   3
    (defun deg (x)
1
2
      ;; TYPE real -> angle
                                                                                   1
                                                                                        (defun latitude (location)
      ;; TYPE list-of-reals -> list-of-angles
3
                                                                                          :: TYPE location -> half-circle
                                                                                   2
4
      ;; x degrees.
                                                                                          (first location))
                                                                                   3
      ;; For typesetting purposes.
5
```

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029

6

x)

1	(defun longitude (location)		1	(defun direction (location focus)	(14.6)
2	;; TYPE location -> circle		2	;; TYPE (location location) -> angle	
3	(second location))		3	;; Angle (clockwise from North) to face focus when	
			4	;; standing in location. Subject to errors near focus and	
			5	;; its antipode.	
			6	(let* ((phi (latitude location))	
1	(defun elevation (location)		7	(phi-prime (latitude focus))	
2	;; TYPE location -> distance		8	(psi (longitude location))	
3	(third location))		9	(psi-prime (longitude focus))	
			10	(y (sin-degrees (- psi-prime psi)))	
			11	(x	
	(defun zone (legation)		12	(- (* (cos-degrees phi)	
1	(defun zone (location)		13	(tan-degrees phi-prime))	
2	<pre>;; TYPE location -> real (fourth location))</pre>		14	(* (sin-degrees phi)	
3	(Tourth Tocation))		15	(cos-degrees	
			16	(- psi psi-prime))))))	
			17	(cond ((or (= x v 0) (= phi-prime (deg 90)))	
1	(defconstant mecca	(14.3)	18	(deg 0))	
2	;; TYPE location		19	((= phi-prime (deg -90))	
3	;; Location of Mecca.		20	(deg 180))	
4	(location (angle 21 25 24) (angle 39 49 24)		21	(t (arctan-degrees v x)))))	
5	(mt 298) (hr 3)))			· · · · · · · · · · · · · · · · · · ·	
				The following functions compute times:	
		(14.4)	1	(defun zone-from-longitude (phi)	(14.8)
1	(derconstant jerusalem	(14.4)	2	;; TYPE circle -> duration	
2	;; TYPE location		3	;; Difference between UT and local mean time at longitude	
3	;; Location of Jerusalem.		4	;; phi as a fraction of a day.	
4	(location (deg 31.78L0) (deg 35.24L0) (mt 740) (nr 2)))		5	(/ phi (deg 360)))	
1	(defconstant acre	(14.5)	1	(defun universal-from-local (tee_ell location)	(14.9)
2	;; TYPE location		2	;; TYPE (moment location) -> moment	. /
3	;; Location of Acre.		3	;; Universal time from local tee_ell at location.	
4	(location (deg 32.94L0) (deg 35.09L0) (mt 22) (hr 2)))		4	(- tee_ell (zone-from-longitude (longitude location))))	

1	(defun local-from-universal (tee_rom-u location)	(14.10)	1	(defun ephemeris-correction (tee)	(14.15)
2	;; TYPE (moment location) -> moment		2	;; TYPE moment -> fraction-of-day	
3	;; Local time from universal tee_rom-u at location.		3	;; Dynamical Time minus Universal Time (in days) for	
4	(+ tee_rom-u (zone-from-longitude (longitude location))))		4	;; moment tee. Adapted from "Astronomical Algorithms"	
			5	;; by Jean Meeus, Willmann-Bell (1991) for years	
			6	;; 1600-1986 and from polynomials on the NASA	
1	(defun standard-from-universal (tee rom-u location)	(14.11)	7	;; Eclipse web site for other years.	
2	·· TYPE (moment location) -> moment	(1.1.1)	8	(let* ((year (gregorian-year-from-fixed (floor tee)))	
3	:: Standard time from <i>tee rom-u</i> in universal time at		9	(c (/ (gregorian-date-difference	
4	;; location.		10	(gregorian-date 1900 january 1)	
5	(+ tee rom-u (zone location)))		11	(gregorian-date year july 1))	
5	(, <u>ccc_</u> rom a (<u>rome rocacrom</u>)))		12	36525))	
			13	(c2051 (* 1/86400	
			14	(+ -20 (* 32 (expt (/ (- year 1820) 100) 2)	i)
1	(defun universal-from-standard (tee_rom-s location)	(14.12)	15	(* 0.5628L0 (- 2150 year)))))	
2	;; TYPE (moment location) -> moment		16	(y2000 (- year 2000))	
3	;; Universal time from <i>tee_rom-s</i> in standard time at		17	(c2006 (* 1/86400	
4	;; location.		18	(poly y2000	
5	(- tee_rom-s (zone location)))		19	(list 62.92L0 0.32217L0 0.005589L0)))))
			20	(c1987 (* 1/86400	
			21	(poly y2000	
1	(defun standard-from-local (tee_ell location)	(14.13)	22	(list 63.86L0 0.3345L0 -0.060374L0	
2	;; TYPE (moment location) -> moment		23	0.0017275L0	
3	;; Standard time from local tee_ell at location.		24	0.000651814L0 0.00002373599L0)))))
4	(standard-from-universal		25	(c1900 (poly c	
5	(universal-from-local tee_ell location)		26	(list -0.00002L0 0.000297L0 0.025184L0	
6	location))		27	-0.181133L0 0.553040L0 -0.861938L0)
			28	0.677066L0 -0.212591L0)))	
			29	(c1800 (poly c	
1	(defun local-from-standard (tee rom-s location)	(14.14)	30	(list -0.000009L0 0.003844L0 0.083563L0	
2	·· TYPE (moment location) => moment	(1.11.1)	31	0.865736L0	
3	: Local time from standard tee rom-s at location		32	4.867575L0 15.845535L0 31.332267L0)
4	(local-from-universal		33	38.291999L0 28.316289L0 11.636204I	0
5	(universal-from-standard tee rom-s location)		34	2.043794L0)))	
6	location))		35	(y1700 (- year 1700))	
-	···· · · · · · · · · · · · · · · · · ·		36	(c1700 (* 1/86400	

(poly y1700	3	;; Dynamical time at Universal moment tee_rom-u.	
(list 8.118780842L0 -0.005092142L0	4	(+ tee_rom-u (ephemeris-correction tee_rom-u)))	
0.003336121L0 -0.0000266484L0))))			
(y1600 (- year 1600))			
(c1600 (* 1/86400	1	(defun universal-from-dynamical (tee)	(14.17)
(poly y1600	2	;; TYPE moment -> moment	
(list 120 -0.9808L0 -0.01532L0	3	;; Universal moment from Dynamical time tee.	
0.000140272128L0))))	4	(- tee (ephemeris-correction tee)))	
(y1000 (/ (- year 1000) 100L0))			
(c500 (* 1/86400			
(poly y1000	1	(defun julian-centuries (tee)	(14.18)
(list 1574.2L0 -556.01L0 71.23472L0 0.319781L0	2	;; TYPE moment -> century	
-0.8503463L0 -0.005050998L0	3	;; Julian centuries since 2000 at moment tee.	
0.0083572073L0))))	4	(/ (- (dynamical-from-universal tee) j2000)	
(y0 (/ year 100L0))	5	36525))	
(c0 (* 1/86400			
(yoly vlog)			
(list 10583.6L0 -1014.41L0 33.78311L0	1	(defconstant j2000	(14.19)
-5.952053L0 -0.1798452L0 0.022174192L0	2	;; TYPE moment	
0.0090316521L0))))	3	;; Noon at start of Gregorian year 2000.	
(v1820 (/ (- vear 1820) 100L0))	4	(+ (hr 12L0) (gregorian-new-year 2000)))	
(other (* 1/86400			
(polv v1820 (list -20 0 32)))))			
(cond ((<= 2051 year 2150) c2051)	1	(defun equation-of-time (tee)	(14.20)
((<= 2006 year 2050) c 2006)	2	;; TYPE moment -> fraction-of-day	
((<= 1987 year 2005) c1987)	3	;; Equation of time (as fraction of day) for moment tee.	
((<= 1900 year 1986) c1900)	4	;; Adapted from "Astronomical Algorithms" by Jean Meeus,	
((<= 1800 year 1899) c1800)	5	;; Willmann-Bell, 2nd edn., 1998, p. 185.	
((<= 1700 year 1799) c1700)	6	(let* ((c (julian-centuries tee))	
((<= 1600 year 1699) c1600)	7	(lambda	
((<= 500 year 1599) c500)	8	(poly c	
((< -500 year 1000))	9	(deg (list 280.46645L0 36000.76983L0	
((tother))))	10	0.0003032L0))))	
	11	(anomaly	
	12	(poly c	
(defun dynamical-from-universal (tee_rom-u) (14.16)	13	(deg (list 357.52910L0 35999.05030L0	
;; TYPE moment -> moment	14	-0.0001559L0 -0.00000048L0))))	

15	(eccentricity		1	(defun universal-from-apparent (tee location)	(14.24)
16	(poly c		2	·· TYPE (moment location) -> moment	(14.24)
17	(ligt 0 01670861710 -0 00004203710		2	. Universal time from sundial time too at location	
19			4	(universal-from-local	
10	(warrangilan (abliguity too))		+	(legg] from apparent too leggtion)	
19	(varepsiton (obrigatcy tee))		5	(iocal-from-apparent tee iocation)	
20	(y (expt (tan-degrees (/ Varepsiion 2)) 2))		0	iocación))	
21	(equation				
22	(* (/ I 2 pI)				
23	(+ (* y (sin-degrees (* 2 lambda)))		1	(defun midnight (date location)	(14.25)
24	(* -2 eccentricity (sin-degrees anomaly))		2	;; TYPE (fixed-date location) -> moment	
25	(* 4 eccentricity y (sin-degrees anomaly)		3	;; Universal time of true (apparent)	
26	(cos-degrees (* 2 lambda)))		4	;; midnight of fixed date at location.	
27	(* -0.5L0 y y (sin-degrees (* 4 lambda)))		5	(universal-from-apparent date location))	
28	(* -1.25L0 eccentricity eccentricity				
29	(sin-degrees (* 2 anomaly)))))))				
30	(* (sign equation) (min (abs equation) (hr 12L0)))))		1	(defun midday (date location)	(14.26)
			2	;; TYPE (fixed-date location) -> moment	
1	(defun apparent-from-local (tee ell location)	(14.21)	3	;; Universal time on fixed date of midday at location.	
2	:: TYPE (moment location) -> moment		4	(universal-from-apparent (+ date (hr 12)) location))	
3	:: Sundial time from local time tee ell at location.				
4	(+ tee ell (equation-of-time				
5	(universal-from-local tee ell location))))		1	(defun sidereal-from-moment (tee)	(14.27)
	(2	:: TYPE moment -> angle	
			3	:: Mean sidereal time of day from moment tee expressed	
1	(defun local-from-apparent (tee location)	(14.22)	4	;; as hour angle. Adapted from "Astronomical Algorithms"	
2	;; TYPE (moment location) -> moment		5	:: by Jean Meeus. Willmann-Bell. Inc., 2nd edn., 1998, p. 88.	
3	;; Local time from sundial time tee at location.		6	(let* ((c (/ (- tee i2000) 36525)))	
4	(- tee (equation-of-time (universal-from-local tee location))))	7	(mod (poly c	
			8	(deg (list 280 4606183710	
1	(defun apparent-from-universal (tee rom-u location)	(14.23)	0	(409 (1150 20011000103720 (+ 36525 360 9856473662910)	
2	·· TYPE (moment location) -> moment	(14.25)	10	$(\times 50525 500.5050475002510)$	
3	True (apparent) time at universal time tee at location		11	360)))	
4	(apparent_from_local		11	500///	
5	(local-from-universal too rem-u location)				
5	(location))				

	Additional solar and lunar astronomical functions are:		1	(defconstant mean-tropical-year	(14.31)
		(14.39)	2	;; TYPE duration	
1	(defun obliquity (tee)	(14.28)	3	365.24218910)	
2	;; TYPE moment -> angle				
3	;; Obliquity of ecliptic at moment tee.				
4	(let* ((c (julian-centuries tee)))		1	(defconstant mean-sidereal-year	(14.32)
5	(+ (angle 23 26 21.448L0)		2	;; TYPE duration	
6	(poly c (list OLO		3	365.25636L0)	
7	(angle 0 0 -46.8150L0)				
8	(angle 0 0 -0.00059L0)				
9	(angle 0 0 0.001813L0))))))		1	(defun solar-longitude (tee)	(14.33)
			2	;; TYPE moment -> season	
		(14.20)	3	;; Longitude of sun at moment tee.	
1	(defun declination (tee beta lambda)	(14.29)	4	;; Adapted from "Planetary Programs and Tables from -4000	
2	;; TYPE (moment half-circle circle) -> angle		5	;; to +2800" by Pierre Bretagnon and Jean-Louis Simon,	
3	;; Declination at moment UT <i>tee</i> of object at		6	;; Willmann-Bell, 1986.	
4	;; latitude beta and longitude lambda.		7	(let* ((c ; moment in Julian centuries	
5	(let* ((varepsilon (obliquity tee)))		8	(julian-centuries tee))	
6	(arcsin-degrees (+ (* (sin-degrees beta)		9	(coefficients	
7	(cos-degrees varepsilon))		10	(list 403406 195207 119433 112392 3891 2819 1721	
8	(* (cos-degrees beta)		11	660 350 334 314 268 242 234 158 132 129 114	
9	(sin-degrees varepsilon)		12	99 93 86 78 72 68 64 46 38 37 32 29 28 27 27	
10	(sin-degrees lambda))))))		13	25 24 21 21 20 18 17 14 13 13 13 12 10 10 10	
			14	10))	
			15	(multipliers	
1	(defun right-ascension (tee beta lambda)	(14.30)	16	(list 0.9287892L0 35999.1376958L0 35999.4089666L0	
2	;; TYPE (moment half-circle circle) -> angle		17	35998.7287385L0 71998.20261L0 71998.4403L0	
3	;; Right ascension at moment UT tee of object at		18	36000.35726L0 71997.4812L0 32964.4678L0	
4	;; latitude beta and longitude lambda.		19	-19.4410L0 445267.1117L0 45036.8840L0 3.1008L0	
5	(let* ((varepsilon (obliquity tee)))		20	22518.4434L0 -19.9739L0 65928.9345L0	
6	(arctan-degrees ; Cannot be bogus		21	9038.0293L0 3034.7684L0 33718.148L0 3034.448L0	
7	(- (* (sin-degrees lambda)		22	-2280.773L0 29929.992L0 31556.493L0 149.588L0	
8	(cos-degrees varepsilon))		23	9037.750L0 107997.405L0 -4444.176L0 151.771L0	
9	(* (tan-degrees beta)		24	67555.316L0 31556.080L0 -4561.540L0	
10	(sin-degrees varepsilon)))		25	107996.706L0 1221.655L0 62894.167L0	
11	(cos-degrees lambda))))		26	31437.369L0 14578.298L0 -31931.757L0	

27	34777.243L0 1221.999L0 62894.511L0		10	(+ (* (deg -0.004778L0) (sin-degrees cap-A))	
28	-4442.039L0 107997.909L0 119.066L0 16859.071L0		11	(* (deg -0.0003667L0) (sin-degrees cap-B)))))	
29	-4.578L0 26895.292L0 -39.127L0 12297.536L0				
30	90073.778L0))				(14.25)
31	(addends		1	(defun aberration (tee)	(14.55)
32	(list 270.54861L0 340.19128L0 63.91854L0 331.26220L0		2	;; TYPE moment -> circle	
33	317.843L0 86.631L0 240.052L0 310.26L0 247.23L0		3	;; Aberration at moment tee.	
34	260.87L0 297.82L0 343.14L0 166.79L0 81.53L0		4	(let* ((c ; moment in Julian centuries	
35	3.50L0 132.75L0 182.95L0 162.03L0 29.8L0		5	(julian-centuries tee)))	
36	266.4L0 249.2L0 157.6L0 257.8L0 185.1L0 69.9L0		6	(- (* (deg 0.0000974L0)	
37	8.0L0 197.1L0 250.4L0 65.3L0 162.7L0 341.5L0		7	(cos-degrees	
38	291.6L0 98.5L0 146.7L0 110.0L0 5.2L0 342.6L0		8	(+ (deg 177.63L0) (* (deg 35999.01848L0) c))))	
39	230.9L0 256.1L0 45.3L0 242.9L0 115.2L0 151.8L0		9	(deg 0.005575L0))))	
40	285.3L0 53.3L0 126.6L0 205.7L0 85.9L0				
41	146.1L0))				
42	(lambda		1	(defun solar-longitude-after (lambda tee)	(14.36)
43	(+ (deg 282.7771834L0)		2	;; TYPE (season moment) -> moment	
44	(* (deg 36000.76953744L0) c)		3	;; Moment UT of the first time at or after <i>tee</i>	
45	(* (deg 0.000005729577951308232L0)		4	;; when the solar longitude will be <i>lambda</i> degrees.	
46	(sigma ((x coefficients)		5	(let* ((rate ; Mean days for 1 degree change.	
47	(y addends)		6	(/ mean-tropical-year (deg 360)))	
48	(z multipliers))		7	(tau ; Estimate (within 5 days).	
49	(* x (sin-degrees (+ y (* z c)))))))))		8	(+ tee	
50	(mod (+ lambda (aberration tee) (nutation tee))		9	(* rate	
51	360)))		10	(mod (- lambda (solar-longitude tee)) 360))))	
			11	(a (max tee (- tau 5))) ; At or after tee.	
			12	(b (+ tau 5)))	
			13	(invert-angular solar-longitude lambda	
1	(defun nutation (tee)	(14.34)	14	(interval-closed a b))))	
2	;; TYPE moment -> circle				
3	;; Longitudinal nutation at moment tee.				
4	(let* ((c ; moment in Julian centuries		1	(defun season-in-gregorian (season g-year)	(14.37)
5	(julian-centuries tee))		2	;; TYPE (season gregorian-year) -> moment	
6	(cap-A (poly c (deg (list 124.90L0 -1934.134L0		3	;; Moment UT of season in Gregorian year g-year.	
7	0.002063L0))))		4	(let* ((jan1 (gregorian-new-year g-year)))	
8	(cap-B (poly c (deg (list 201.11L0 72001.5377L0		5	(solar-longitude-after season jan1)))	
9	0.00057L0)))))				

(14.42)

$\begin{array}{cccccccccccccccccccccccccccccccccccc$	grees, ignoring
3 ;; Precession at moment tee using 0,0 as J2000 coordinates. 5 ;; parallax and refraction. 4 ;; Adapted from "Astronomical Algorithms" by Jean Meeus, 6 (let* ((phi ; Local latitude. 5 ;; Willmann-Bell, 2nd edn., 1998, pp. 136-137. 7 (latitude location)) 6 (let* ((c (julian-centuries tee)) 8 (psi ; Local longitude. 7 (eta (mod 9 (longitude location)) 8 (poly c (list 0 (secs 47.0029L0) 10 (lambda ; Solar longitude. 9 (poly c (list 0 (secs 47.0029L0) 11 (solar-longitude tee)) 10 (secs -0.03302L0) 11 (solar-longitude tee)) 11 360)) 12 (alpha ; Solar right ascension 12 (cap-P (mod (poly c (list (deg 174.876384L0) 14 (delta ; Solar declination. 13 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 14 (secs 1.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees cap-P)))) 21 (cap-B (cos-degrees cap-P)) 23 <td></td>	
4 ;; Adapted from "Astronomical Algorithms" by Jean Meeus, 6 (let* ((phi; Local latitude. 5 ;; Willmann-Bell, 2nd edn., 1998, pp. 136-137. 7 (latitude location)) 6 (let* ((c (julian-centuries tee)) 8 (psi; Local longitude. 7 (eta (mod 9 (longitude location)) 8 (poly c (list 0 (secs 47.0029L0) 10 (lambda; Solar longitude. 9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.00006DL0))) 12 (alpha; Solar right ascension 11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0) 14 (delta; Solar declination. 13 (secs 0.03536L0))) 16 (theta0; Sidereal time. 14 (secs 0.03536L0))) 16 (theta0; -psi) alpha) 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0) 18 (cap-H ; Local hour angle. 17 (secs 0.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.000006L0))) 20 (altitude	
5 ;; Willmann-Bell, 2nd edn., 1998, pp. 136-137. 7 (latitude location)) 6 (let* ((c (julian-centuries tee)) 8 (psi ; Local longitude. 7 (eta (mod 9 (longitude location)) 8 (poly c (list 0 (secs 47.0029L0) 10 (lambda ; Solar longitude. 9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.00060L0))) 12 (alpha ; Solar right ascension 11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda)) 14 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0) 18 (cap-H ; Local hour angle. 17 (secs 0.11113L0) 19 (mod (- theta0 (- ps) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (ap-P)))) 20	
6 (let* ((c (julian-centuries tee)) 8 (psi ; Local longitude. 7 (eta (mod 9 (longitude location)) 8 (poly c (list 0 (secs 47.0029L0)) 10 (lambda ; Solar longitude. 9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.00006D0))) 12 (alpha ; Solar right ascension 11 360) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda) 14 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0) 18 (cap-H ; Local hour angle. 17 (secs 0.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees cap-P)))) 23 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees cap-P))	
7 (eta (mod 9 (longitude location)) 8 (poly c (list 0 (secs 47.0029L0) 10 (lambda ; Solar longitude. 9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.00006D0))) 12 (alpha ; Solar right ascension 11 360) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda) 14 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0) 18 (cap-H ; Local hour angle. 17 (secs 1.11113L0) 19 (mod (- theta0 (- ps)) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees cap-P)))) 22 21 (cap-B (cos-degrees cap-P))) 23 (* (cos-degrees cap-P))	
8 (poly c (list 0 (secs 47.0029L0)) 10 (lambda ; Solar longitude. 9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.000060L0))) 12 (alpha ; Solar right ascension 11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda) 14 (secs -869.8089L0) 16 (theta0 ; Sideral time. 15 360)) 16 (theta0 ; Sideral time. 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 0.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (soledegrees (- p-P))))) 23 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees (- (* (cos-degrees (- (* (sin-degrees (- (* (sin-degrees (- (* (sin-degrees (- (* (sin-degr	
9 (secs -0.03302L0) 11 (solar-longitude tee)) 10 (secs 0.000060L0))) 12 (alpha ; Solar right ascension 11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda) 14 (secs 0.03536L0)) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 0.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (-ap-P))))) 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees (-ap-P))	
10 (secs 0.00060L0))) 12 (alpha; Solar right ascension 11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta; Solar declination. 13 (secs -869.8089L0) 14 (delta; Solar declination. 14 (secs 0.03536L0))) 16 (theta0; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.0006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (- ap-P))))) 21 (cap-A (* (cos-degrees eta) (sin-degrees cap-P))) 23 (* (cos-degrees (- ap-P))	
11 360)) 13 (right-ascension tee 0 lambda) 12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda) 14 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (-ap-P)))))) 20 (cap-A (* (cos-degrees eta) (sin-degrees cap-P))) 23 (* (cos-degrees (-ap-P))	1.
12 (cap-P (mod (poly c (list (deg 174.876384L0)) 14 (delta ; Solar declination. 13 (secs -869.8089L0) 15 (declination tee 0 lambda)) 14 (secs 0.03536L0))) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (- ap-P)))))) 20 (cap-A (* (cos-degrees eta) (sin-degrees cap-P))) 23 (* (cos-degrees degrees degree	ı))
13 (secs -869.8089L0) 15 (declination tee 0 lambda)) 14 (secs 0.03536L0)) 16 (theta0 ; Sidereal time. 15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.11113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (son-degrees cap-P)))) 22 (sin-degrees (+ (* (cos-degrees cap-P))) 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees cap-P))	
14 (secs 0.03536L0)) 16 (theta0; Sidereal time. 15 360) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.1113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (- (sin-degre	
15 360)) 17 (sidereal-from-moment tee)) 16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.1113L0) 19 (mod (- theta0 (- psi) alpha)) 18 (secs 0.000006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (- processed)))))) 22 (sin-degrees (+ (* (cos-degrees eta) (sin-degrees cap-P))) 23 (* (cos-degrees eta))	
16 (p (mod (poly c (list 0 (secs 5029.0966L0)) 18 (cap-H ; Local hour angle. 17 (secs 1.1113L0) 19 (mod (- theta0 (- psi) alpha)) 18 (secs 0.000006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (- product))))))))))))))))))))))))))))))))))))	
17 (secs 1.1113L0) 19 (mod (- theta0 (- psi) alpha) 18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (* (cos-degrees cap-P))) 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees (+ (* (cos-degrees (+ (* (cos-degrees (- (- (cos-degrees (- (cos-degrees (- (cos-degrees (- (- (cos-degrees (- (cos-de	
18 (secs 0.00006L0))) 20 (altitude 19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (sin-degrees (+ (* (sin-degrees (+ (si-	360))
19 360)) 21 (arcsin-degrees (+ (* (sin-degrees (+ (sin-degrees (+ (* (sin-degrees (+ (* (sin-degrees (+ (si (sin-degrees (+ (si) (sin degrees (+ (si-	
20 (cap-A (* (cos-degrees eta) (sin-degrees cap-P))) 22 (sin-degrees cap-P)) 21 (cap-B (cos-degrees cap-P)) 23 (* (cos-degrees cap-P))	grees phi)
21 (cap-B (cos-degrees cap-P)) 23 (* (cos-de	grees delta))
	grees phi)
22 (arg (arctan-degrees cap-A cap-B))) 24 (cos-de	grees delta)
23 (mod (- (+ p cap-P) arg) 360))) 25 (cos-de	grees cap-H))))))
26 (mod3 altitude -180 180)))	
1 (defun sidereal-solar-longitude (tee) (14.40)	
2 ;; TYPE moment -> angle	(lambda tee)
3 ;; Sidereal solar longitude at moment tee 2 TYPE (season moment) => moment	Tumbuu CCC)
4 (mod (+ (solar-longitude tee)	20
5 (- (precession tee)) 4 ·· when solar longitude just even define	lambda degrees
6 sidereal-start) 5 (let+ (rate - Mean change of one der	ree
7 360)) 6 (/ mean-tronical-year (deg 3	50)))
7 (tau · First approximation	
8 (- tee	
1 (defun solar-altitude (tee location) (14.41) 9 (* rate (mod (- (solar-lor	gitude tee)
2 ;; TYPE (moment location) -> half-circle 10 lambda)	

12 (cap-bla ; Difference in longitude. 26 (moon-argument ; Moon's argument of latitude. 13 (mod) (- (solar-longitude tau) lambda) 27 (poly c (deg (list 16.7106U.0 (* 390.67050284D0 14 -180 180))) 28 -0.0016118L0 -0.0000027D0 15 (min tee (- tau (* rate cap-Delta)))) 29 -0.0000001100))) 15 (min tee (- tau (* rate cap-Delta)))) 20 -0.00000027D0 16 (cap-comega ; Longitude of azeending node. (0.0020672D0 0.0000021D1))) 17 (defunnth-new-moon (n) (H444) 3 0.0020672D0 0.0000021S1D1 12 (j rTFE integer -> moment (H445) (solar-coeff (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 2 0 0 0 2 1 2 12 (j rTFE integer -> moment (Iunar-coeff (list 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0 2 - 2 0 0 1 0 1 1 1 2 0 1 0 0 2 1 1 1 3 0) 13 (j rof Jonuary 11, I. Adapted from 'Astrononical Algorithms' 4 (moon-coeff (list 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0 2 - 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 0	11	360))))		25	-0.00000058L0))))
immedic (nolar-longitude taw) lambda) 2 (poly c (deg (list 160.7106L) (.390.67050284L) immedic -180 180)) 1236.85L0) 1236.85L0) immedic (nin tee (- taw (* rate cap-Delta))))) 0.0000001LD)))) 1236.85L0) immedic (cap-comega / Longitude of ascending node. 0.00000221L0 (.0000021L0))) immedic (ifefconstant mean-synodic-month) (144) 3 (poly c (deg (list 124.7746L0 (* - 1.5637558EL0 1236.85L0)) immedic (ifefun nth-new-moon (n) (144) 3 (poly c (deg (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 - 0 0 0 0 immedic (ifefun nth-new-moon (n) (144) 3 (unar-coeff (list 0 1 0 0 - 1 1 2 0 0 1 0 1 1 - 1 - 1 0)) immedic immedic immedic immedic immedic immedic immedic immedic immedic	12	(cap-Delta ; Difference in longitude.		26	(moon-argument ; Moon's argument of latitude.
14 -180 180))) 28 1236.85L0) 15 (min tee (- tau (+ rate cap-Delta)))) 29 -0.0016118L0 -0.0000027L0 15 (defconstant mean-synodic-month (144) 3 (cap-cmega ; Longitude of accenting node. 2 ;; TYPE duration 3 (defconstant mean-synodic-month (144) 3 (poly c (deg (lit 124.7746L0 + -1.5675588L0 1236.85L0) 2 ;; SOSS8861L0) 4 (E-factor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0) 3 29.530588861L0) 4 (E-factor (list 0 1 0 0 - 1 1 2 0 0 1 0 1 1 1 - 1 0 0 0 0 1 2 0 0 0 - 2 1 0 0 - 1 1 2 0 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 1 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 0 0 0 - 2 1 0 0 - 2 0 0 0 - 2 1 0 0 - 0 - 0 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 - 2 0 0 0 -	13	(mod3 (- (solar-longitude tau) lambda)		27	(poly c (deg (list 160.7108L0 (* 390.67050284L0
15 (min tee (- tau (+ rate cap-Delta)))) 29 -0.0016118L0 -0.0000227L0 10 (defconstant mean-synodic-month (144) 31 (cap-omega ; Longitude of ascending node. 11 (defconstant mean-synodic-month (144) 32 (poly c (deg (list 124.7746L0 (+ -1.5637558L0 1236.85L0))) 2 :: TYPE duration 33 0.00020672L0.0000021L0))) 3 29.53058861L0 0.0 0 0 0 0 1 1 1 0 0 1 1 1 0 0 0 0 3 29.53058861L0 0.0 0 0 0 0 0 0 0) 3 29.53058861L0 10 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 4 :: TYPE integer -> moment (solar-coeff (list 0 1 0 0 - 1 1 2 0 0 1 0 1 1 - 1 2 0 0 0 - 2 2 0 2 - 2 0 0 0 5 :: JYDE integer -> moment (unar-coeff (list 0 0 1 0 0 - 1 2 0 0 0 - 2 - 2 0 0 - 2 - 2 0 0 0 - 2 - 2	14	-180 180)))		28	1236.85L0)
i 0 0.0000001L0()))) i (cag-omega; Longitude of ascending node. i (poly c (deg (list 124.7746L0 (t - 1.56375588L0 1236.85L0)) i (poly c (deg (list 124.7746L0 (t - 1.56375588L0 1236.85L0)) i (poly c (deg (list 124.7746L0 (t - 1.56375588L0 1236.85L0)) i (poly c (deg (list 10 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 i (fefactor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 i (defun nth-new-moon (n) (H445) i (solar-coeff (list 1 0 2 0 1 1 0 1 1 2 0 0 0 2 2 0 0 2 - 2 0 0 i ; forganary 11, 1. Adayted from "Astronomical Algorithms" (moon-coeff (list 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0 i ; j: by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. (moon-coeff (list 0 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0 i ; j: by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. (moon-coeff (list 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	15	(min tee (- tau (* rate cap-Delta)))))		29	-0.0016118L0 -0.00000227L0
1 (der_constant mean-synchic - month (14.44) 12 (poly c (deg (list 124.7746L0 (+ -1.5637588L0 126.65L0)) 2 ;; TYPE duration 3 0.0020672L0 0.0000015L0)))) 3 29.530588661L0) 4 (E-factor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 0 0				30	0.00000011L0))))
1 (defconstant mean-synodic-month (144) 32 (poly c (deg (list 124.7746.0 (* -1.56375584.0 1236.85.0))) 2 ;; TYPE duration 33 0.0020672L0 0.000000215L0)))) 3 29.530588610) (E-factor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 0 0				31	(cap-omega ; Longitude of ascending node.
2 ;; TYPE duration 33 0.0020672L0 0.00000215L0)))) 3 29.530588861L0) 34 (E-factor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0 3 29.530588861L0) 34 (E-factor (list 0 1 0 0 - 1 1 2 0 0 1 0 1 1 1 - 1 2 1 (defun nth-new-moon (n) (l445) 37 0 3 1 0 1 - 1 - 1 0) 2 ;; TYPE integer -> moment 39 0 1 2 2 1 1 1 3 4) 30 3 ;; Moment of n-th new moon after (or before) the new moon 40 (moon-coeff (list 0 0 0 - 0 0 0 - 2 2 0 0 2 - 2 0 0 2 - 2 0 0 2 - 2 0 0 2 - 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 2 2 0 0 0 - 0 0 0 0	1	(defconstant mean-synodic-month	(14.44)	32	(poly c (deg (list 124.7746L0 (* -1.56375588L0 1236.85L0)
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	2	;; TYPE duration		33	0.0020672L0 0.00000215L0))))
35 0 0 0 0 0 0 0) 1 (defun nth-new-moon (n) (1445) 3 0 1 0 1 - 1 - 1 0) 2 ;; TYPE integer -> moment 0 1 2 0 1 0 0 1 - 1 2 3 0 0 2 1 2 3 ;; Moment of n-th new moon after (or before) the new moon 0 (moon-coeff (list 0 0 0 2 0 0 0 - 2 0 0 2 - 2 0 0 2 - 2 0 0) 4 ;; of January 11, 1. Adapted from *Astronomical Algorithms* 4 -2 0 - 2 2 2 2 - 2 0 0) 5 ; by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. 2 (sine-coeff 6 (let* ((n0 24724) ; Months from RD 0 until j2000. 4 0.00735L0 - 0.0051AL0 0.00208L0 7 (k (- n n0)); Months since j2000. 4 0.00735L0 - 0.0003L0 0.01038L0 8 (c / k 1236.85L0) ; Juliar centuries. 4 0.0003L0 0.0003L0 0.00038L0 9 (approx (+ j2000) 4 -0.00042L0 0.0003L0 0.00038L0 10 (poly c (list 5.09766L0 4 -0.00042L0 0.0003L0 0.00003L0 12 1236.85L0 4 -0.00032L0 0.0003L0 13 0.000150L0 (carbet (nol 0 1 0.0003L0 - 0.00003L0 -0.00003L0 0.00003L0 14 0.0000150L0 (carbet (nol 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	3	29.530588861L0)		34	(E-factor (list 0 1 0 0 1 1 2 0 0 1 0 1 1 1 0 0 0 0
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$				35	0 0 0 0 0 0))
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$				36	(solar-coeff (list 0 1 0 0 -1 1 2 0 0 1 0 1 1 -1 2
1 (definit infinitements) 38 (lunar-coeff (list 1 0 2 0 1 1 0 1 1 2 3 0 0 2 1 2 0 1 2 1 1 1 3 4)) 3 ;; TYPE integer -> moment 39 0 1 2 1 1 1 3 4)) 4 ;; of January 11, 1. Adapted from "Astronomical Algorithms" 40 (moon-coeff (list 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0)) 4 ;; of January 11, 1. Adapted from "Astronomical Algorithms" 41 -2 0 - 2 2 2 2 2 - 2 0 0)) 5 ;; by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. 42 (sine-coeff 6 (let* ((no 24724); Months from RD 0 until j2000. 43 (list - 0.40720L0 0.17241L0 0.01608L0 0.01039L0 7 (k (-n n0)); Months since j2000. 44 0.00739L0 -0.00514L0 0.00005L0 0.00038L0 8 (c (/ k 1236.85L0)); julian centuries. 45 -0.00111L0 -0.00005L0 0.00038L0 10 (poly c (list 5.09766L0 47 -0.00024L0 -0.00003L0 0.00038L0 11 (* mean-synodic-month 48 0.00003L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00002L0 0.00003L0 0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0 0.00002L0 14 -0.000000150.01 51 ((14.45)	37	0 3 1 0 1 -1 -1 1 0))
2 ;; TTPE integer -> moment 39 0 1 2 1 1 1 3 4)) 3; ;; Moment of n-th new moon after (or before) the new moon 4 (moon-coeff (list 0 0 0 2 0 0 0 - 2 2 0 0 2 - 2 0 0) 4 ;; of January 11, 1. Adapted from *Astronomical Algorithms* 4 -2 0 - 2 2 2 2 2 - 2 0 0)) 5 ;; by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. 42 (sine-coeff 6 (let* ((n0 24724); Months from RD 0 until j2000. 43 (list -0.40720L0 0.1724L0 0.00608L0 0.01039L0 8 (c (/ k 1236.85L0)); Julian centuries. 44 0.00739L0 -0.00514L0 0.00028L0 9 (approx (+ j200) 6 -0.0011L0 -0.000057L0 0.000056L0 10 (poly c (list 5.09766L0 47 -0.00024L0 0.00003L0 0.00003L0 11 (* mean-symodic-month 48 0.00004L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00003L0 0.00003L0 13 0.00015437L0 50 -0.00003L0 0.00003L0 14 -0.00000150L0 50 -0.000001L0 (sin-degrees cap-omega)) 15 0.00000000073L0))) 52 (+ (* -0.0017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coef	1	(derun ntn-new-moon (n)	(14.45)	38	(lunar-coeff (list 1 0 2 0 1 1 0 1 1 2 3 0 0 2 1 2
3 ;; proment of n=h finew moon after (of before) the woon and specified of n=w moon after (of before) the woon	2	;; TYPE Integer -> moment		39	0 1 2 1 1 1 3 4))
4 ;; of January 11, 1. Adapted from "Astronomical Argorithms" 41 -2 0 -2 2 2 2 2 -2 0 0)) 5 ;; by Jean Meeus, Willmann-Bell, corrected 2nd edn., 2005. 42 (sine-coeff 6 (let* ((n0 24724); Months from RD 0 until j2000. 43 (list -0.40720L0 0.1724LL0 0.01608L0 0.01039L0 7 (k (-n n0)); Months since j2000. 44 0.00739L0 -0.0051AL0 0.00028L0 8 (c (/ k 1236.85L0)); Julian centuries. 45 -0.0011L0 -0.00057L0 0.0003E0 9 (approx (+ j2000 47 -0.00024L0 -0.00001L0 0.00038L0 10 (velop's clist 5.09766L0 47 -0.00024L0 -0.00003L0 0.00003L0 11 (* mean-synodic-month 48 0.00004L0 0.00003L0 0.00003L0 12 1236.85L0 49 -0.00020L0 0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0 14 -0.00000000073L0)))) 51 (correction 15 0.0000000073L0)))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coeff) 16 (cap-E (poly c (list 2.5534L0 55 (x solar-coeff) 17	3	;; Moment of n-th new moon after (or before) the new moon		40	(moon-coeff (list 0 0 0 2 0 0 0 -2 2 0 0 2 -2 0 0
3 ;; by betain needs, withinkellingent, 2003. 42 (sine-coeff 6 (let* ((n0 24724); Months from RD 0 until j2000. 43 (list -0.40720L0 0.1724LD 0.01608L0 0.01039L0 7 (k (-n n0)); Months since j2000. 44 0.00739L0 -0.00514L0 0.00056L0 8 (c (/k 1236.85L0)); Julian centuries. 45 -0.0011LD -0.00057L0 0.0003EL0 9 (approx (+ j200) 46 -0.00042L0 0.00042L0 0.0003EL0 10 (* mean-synodic-month 48 0.00003L0 0.00003L0 0.00003L0 11 (* mean-synodic-month 48 0.00003L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00002L0 0.0003L0 0.00003L0 13 0.00015437L0 50 -0.00002L0 0.00002L0) 14 -0.000001505D 51 (correction 15 0.0000000073L0)))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) (v solar-coeff) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0)	4	;; of January II, I. Adapted from "Astronomical Algorithms"		41	-2 0 -2 2 2 2 -2 0 0))
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	5	(lot. ((n0.24724) . Months from RD 0 until j2000		42	(sine-coeff
// (k (* 1 h 10)) ; Moltin's since j2000. 44 0.00739L0 -0.00514L0 0.0028L0 8 (c (/ k 1236.85L0)) ; Julian centuries. 45 -0.00111L0 -0.00057L0 0.00056L0 9 (approx (+ j200) 46 -0.00042L0 0.00042L0 0.00038L0 10 (poly c (list 5.09766L0 47 -0.00024L0 -0.00007L0 0.00003L0 11 (* mean-synodic-month 48 0.00034L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00002L0 0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0 14 -0.000000073L0))) 50 -0.00002L0 0.00002L0 15 0.0000000073L0))) 51 (correction 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma (lv sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.0000014L0 -0.00000011L0)))) 57 (z moon-coeff)	7	$(100 \times 4/24)$; Months from RD 0 until j2000.		43	(list -0.40720L0 0.17241L0 0.01608L0 0.01039L0
* (c (/ x 1230.0310)) / 01111 Centuries. 45 -0.00111L0 -0.00057L0 0.00056L0 9 (approx (+ j2000 46 -0.00042L0 0.00042L0 0.00038L0 10 (poly c (list 5.09766L0 47 -0.00024L0 -0.00007L0 0.00004L0 11 (* mean-synodic-month 48 0.00004L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00003L0 0.00003L0 -0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0)) 14 -0.0000000073L0))) 51 (correction 15 0.0000000073L0))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega))) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 21 (lunar-anomaly 57 (z moon-coeff) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	,	$(\mathbf{k} (= 1110))$; Month's since j2000.		44	0.00739L0 -0.00514L0 0.00208L0
9 (applot (*) 2000 46 -0.00042L0 0.00042L0 0.00038L0 10 (poly c (list 5.09766L0 47 -0.00024L0 -0.00007L0 0.00004L0 11 (* mean-synodic-month 48 0.00004L0 0.00003L0 -0.00002L0 12 1236.85L0) 49 -0.00002L0 0.00003L0 -0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0) 14 -0.0000000073L0))) 52 (+ (* -0.0017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.000014L0 -0.0000011L0)))) 57 (z moon-coeff) 21 (lunar-anomaly 58 (* v (expt cap-E w) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	0	(C (/ K 1250.05L0)); Julian Centuries.		45	-0.00111L0 -0.00057L0 0.00056L0
10 () bit y c (11st 3.037800 47 -0.00024L0 -0.00007L0 0.00004L0 11 (* mean-synodic-month 48 0.00004L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00003L0 0.00003L0 -0.00002L0 13 0.0015437L0 50 -0.00002L0 0.00002L0) 14 -0.0000000073L0))) 51 (correction 15 0.000000073L0))) 52 (+ (* -0.0017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 20 -0.000014L0 -0.0000011L0))) 56 (y lunar-coeff) 21 (lunar-anomaly 57 (z moon-coeff) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 58 (* v (expt cap-E w) 23 1236.85L0) 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	10	(approx (+ j2000		46	-0.00042L0 0.00042L0 0.00038L0
11 (* mean-synotic -month 48 0.00004L0 0.00003L0 0.00003L0 12 1236.85L0) 49 -0.00003L0 0.00003L0 -0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0)) 14 -0.00000000073L0))) 51 (correction 15 0.000000074L0))) 52 (+ (* -0.0007L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 20 -0.000014L0 -0.0000011L0))) 56 (y lunar-coeff) 21 (lunar-anomaly 57 (z moon-coeff) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 58 (* v (expt cap-E w)) 23 1236.85L0) 59 (sin-degrees	10	(pory C (rist 5.09766L0		47	-0.00024L0 -0.00007L0 0.00004L0
12 1250.8500 49 -0.00003L0 0.00003L0 -0.00002L0 13 0.00015437L0 50 -0.00002L0 0.00002L0)) 14 -0.000000150L0 51 (correction 15 0.000000073L0)))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 20 -0.000014L0 -0.0000011L0))) 56 (y lunar-coeff) 21 (lunar-anomaly 57 (z moon-coeff)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 59 (solar-anomaly)	11	(* mean-synodic-month		48	0.00004L0 0.00003L0 0.00003L0
13 0.0001437L0 50 -0.00002L0 0.00002L0)) 14 -0.00000150L0 51 (correction 15 0.0000000073L0)))) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 20 -0.0000014L0 -0.0000011L0)))) 56 (y lunar-coeff) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	12	1236.8510)		49	-0.00003L0 0.00003L0 -0.00002L0
14 -0.0000015010 \$1 (correction 15 0.000000007310)))) \$2 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.000074L0))) \$3 (sigma ((v sine-coeff) 17 (solar-anomaly \$4 (w E-factor) 18 (poly c (deg (list 2.5534L0 \$5 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) \$6 (y lunar-coeff) 20 -0.0000014L0 -0.0000011L0)))) \$7 (z moon-coeff)) 21 (lunar-anomaly \$8 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 \$9 (sin-degrees 23 1236.85L0) \$60 (+ (* x solar-anomaly)	15	0.0001543710		50	-0.00002L0 0.00002L0))
15 0.0000000073L0())) 52 (+ (* -0.00017L0 (sin-degrees cap-omega)) 16 (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.000014L0 -0.0000011L0)))) 57 (z moon-coeff)) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0) (* 385.81693528L0) 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	14	-0.00000013010		51	(correction
16 (cap-E (p)ry c (list 1 - 0.002516L0 - 0.000074L0))) 53 (sigma ((v sine-coeff) 17 (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.000014L0 - 0.0000011L0)))) 57 (z moon-coeff)) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	15	0.00000000/3E0/)))		52	(+ (* -0.00017L0 (sin-degrees cap-omega))
1/ (solar-anomaly 54 (w E-factor) 18 (poly c (deg (list 2.5534L0 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.0000014L0 -0.0000001L0)))) 57 (z moon-coeff)) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly))	10	(cap-E (pory C (IISt I -0.002516L0 -0.00000/4L0)))		53	(sigma ((v sine-coeff)
18 (poly c (deg (list 2.55340) 55 (x solar-coeff) 19 (* 1236.85L0 29.10535670L0) 56 (y lunar-coeff) 20 -0.0000014L0 -0.0000001L0)))) 57 (z moon-coeff)) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0) 59 (sin-degrees 23 1236.85L0) 50 (+ (* x solar-anomaly))	17	(solar-anomaly		54	(w E-factor)
19 (* 1230.0510 29.103307010) 56 (y lunar-coeff) 20 -0.000014L0 -0.000001100))) 57 (z moon-coeff) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	18	(poly c (deg (list 2.5534L0		55	(x solar-coeff)
21 (lunar-anomaly 57 (z moon-coeff)) 21 (lunar-anomaly 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly))	20	(* 1230.0510 29.105550/010)		56	(y lunar-coeff)
21 (101a1-allollaly) 58 (* v (expt cap-E w)) 22 (poly c (deg (list 201.5643L0 (* 385.81693528L0 59 (sin-degrees)) 23 1236.85L0) 60 (+ (* x solar-anomaly))	20	-0.000001410 -0.000001110////		57	(z moon-coeff))
22 (pory c (deg (fist 201.5043L0 (* 365.0109526L0 59 (sin-degrees 23 1236.85L0) 60 (+ (* x solar-anomaly)	21	(Iunar-anomary		58	(* v (expt cap-E w)
23 1230.00LU) 60 (+ (* x solar-anomaly)	22	(pory C (deg (iist 201.004)it) (* 380.81693528L0		59	(sin-degrees
	23			60	(+ (* x solar-anomaly)

61	(* y lunar-anomaly)		5	(phi (lunar-phase tee))	
62	(* z moon-argument))))))))		6	(n (round (- (/ (- tee t0) mean-synodic-month)	
63	(add-const		7	(/ phi (deg 360))))))	
64	(list 251.88L0 251.83L0 349.42L0 84.66L0		8	<pre>(nth-new-moon (final k (1- n) (< (nth-new-moon k) tee)))))</pre>	
65	141.74L0 207.14L0 154.84L0 34.52L0 207.19L0				
66	291.34L0 161.72L0 239.56L0 331.55L0))				
67	(add-coeff		1	(defun new-moon-at-or-after (tee)	(14.47)
68	(list 0.016321L0 26.651886L0		2	;; TYPE moment -> moment	
69	36.412478L0 18.206239L0 53.303771L0		3	;; Moment UT of first new moon at or after tee.	
70	2.453732L0 7.306860L0 27.261239L0 0.121824L0		4	(let* ((t0 (nth-new-moon 0))	
71	1.844379L0 24.198154L0 25.513099L0		5	(phi (lunar-phase tee))	
72	3.592518L0))		6	(n (round (- (/ (- tee t0) mean-synodic-month)	
73	(add-factor		7	(/ phi (deg 360)))))	
74	(list 0.000165L0 0.000164L0 0.000126L0		8	(nth-new-moon (next k n (>= (nth-new-moon k) tee)))))	
75	0.000110L0 0.000062L0 0.000060L0 0.000056L0				
76	0.000047L0 0.000042L0 0.000040L0 0.000037L0				
77	0.000035L0 0.000023L0))		1	(defun lunar-longitude (tee)	(14.48)
78	(extra		2	(defun funat-fongicude (cee)	(14.40)
79	(* 0.000325L0		2	;; IIPE moment -> angle	
80	(sin-degrees		3	;; Longitude of moon (in degrees) at moment tee.	
81	(poly c		4	;; Adapted from Astronomical Argorithms by Jean Meeus,	
82	(deg (list 299.77L0 132.8475848L0		5	(lett (/g (iulian conturies too))	
83	-0.009173L0))))))		7	(ret* ((C (juiran-centuries tee))	
84	(additional		,	(cap D (lupar elemention c))	
85	(sigma ((i add-const)		0	(cap-M (colar-aromaly c))	
86	(j add-coeff)		10	(cap-M_primo (lupar_anomaly c))	
87	(l add-factor))		10	(cap-M-prime (runar-anomary c))	
88	(* l (sin-degrees (+ i (* j k))))))		12	(cap - F (moon - node C))	
89	(universal-from-dynamical		12	(cap-E (pory C (fist 1 =0.002510E0 =0.0000074E0)))	
90	(+ approx correction extra additional))))		14	(args-runar-erongation)	
			14		
			15		
	(defum nor mean before (tee)	(14.46)	10		
1	(deful new-moon-before (lee)	(14.40)	19	(args-sotar-anomary)	
2	Moment III of last new moon before tee		10		
3	(latt ((t0 (pth per meen 0)))		19		
4	(rec, ((co (ncn-new-moon o))		20		

(args-lunar-anomaly	57	(+ (deg 53.09L0)	
(list 1 -1 0 2 0 0 -2 -1 1 0 -1 0 1 0 1 1 -1 3 -2	58	(* c (deg 479264.29L0))))))	
-1 0 -1 0 1 2 0 -3 -2 -1 -2 1 0 2 0 -1 1 0	59	(flat-earth	
-1 2 -1 1 -2 -1 -1 -2 0 1 4 0 -2 0 2 1 -2 -3	60	(* (deg 1962/1000000)	
2 1 -1 3))	61	(sin-degrees (- cap-L-prime cap-F)))))	
(args-moon-node	62	(mod (+ cap-L-prime correction venus jupiter flat-earth	
(list 0 0 0 0 0 2 0 0 0 0 0 0 0 -2 2 -2 0 0 0 0	63	(nutation tee))	
0 0 0 0 0 0 2 0 0 0 0 0 -2 2 0 2 0 0 0 0	64	360)))	
0 0 -2 0 0 0 0 -2 -2 0 0 0 0 0 0 0))			
(sine-coeff	1	(defun mean-lunar-longitude (c)	(14.49)
(list 6288774 1274027 658314 213618 -185116 -114332	2	:: TYPE century -> angle	()
58793 57066 53322 45758 -40923 -34720 -30383	3	:: Mean longitude of moon (in degrees) at moment	
15327 -12528 10980 10675 10034 8548 -7888	4	;; given in Julian centuries c.	
-6766 -5163 4987 4036 3994 3861 3665 -2689	5	:: Adapted from "Astronomical Algorithms" by Jean Meeus,	
-2602 2390 -2348 2236 -2120 -2069 2048 -1773	6	:: Willmann-Bell, 2nd edn., 1998, pp. 337-340.	
-1595 1215 -1110 -892 -810 759 -713 -700 691	7	(mod	
596 549 537 520 -487 -399 -381 351 -340 330	8	(poly c	
327 -323 299 294))	9	(deg (list 218.3164477L0 481267.88123421L0	
(correction	10	-0.0015786L0 1/538841 -1/65194000)))	
(* (deg 1/1000000)	11	360))	
(sigma ((v sine-coeff)			
(w args-lunar-elongation)	1	(defun lunar-elongation (c)	(14.50)
(x args-solar-anomaly)	2	·· TYPE century -> angle	(14.50)
(y args-lunar-anomaly)	3	Elongation of moon (in degrees) at moment	
(z args-moon-node))	4	;; given in Julian centuries c	
(* v (expt cap-E (abs x))	5	Adapted from "Astronomical Algorithms" by Jean Meeus	
(sin-degrees	6	 Willmann-Bell 2nd edn 1998 n 338 	
(+ (* w cap-D)	7	(mod	
(* x cap-M)	8	(nolv c	
(* y cap-M-prime)	9	(dem (list 297 850192110 445267 111403410	
(* z cap-F))))))))	10	-0.00188191.0.1/545868 -1/113065000)))	
(venus (* (deg 3958/100000))	11	360))	
(sin-degrees		,,	
(+ (deg 119.75L0) (* c (deg 131.849L0))))))	,	(defun celer enemaly (c)	(14.51)
(jupiter (* (deg 318/100000))	1	(defuil Solar anomaly (C)	(14.31)
(sin-degrees	2	;; IIFE CENCULY -> angle	
	3	;; mean anomaly of sun (in degrees) at moment	

4	;; given in Julian centuries c.
5	;; Adapted from "Astronomical Algorithms" by Jean Meeus,
6	;; Willmann-Bell, 2nd edn., 1998, p. 338.
7	(mod
8	(poly c
9	(deg (list 357.5291092L0 35999.0502909L0
10	-0.0001536L0 1/24490000)))
11	360))

;; Mean anomaly of moon (in degrees) at moment

;; Adapted from "Astronomical Algorithms" by Jean Meeus,

(defun lunar-node (date)	(14.54)
;; TYPE fixed-date -> angle	
;; Angular distance of the lunar node from the equinoctial	
;; point on fixed date.	
<pre>(mod3 (+ (moon-node (julian-centuries date)))</pre>	
-90 90))	

1	(defun sidereal-lunar-longitude (tee)	(14.55)
2	;; TYPE moment -> angle	
3	;; Sidereal lunar longitude at moment tee.	
4	(mod (+ (lunar-longitude tee)	
5	(- (precession tee))	
6	sidereal-start)	
7	360))	

;; Willmann-Bell, 2nd edn., 1998, p. 338.				
(mod				
(poly c		1	(defun lunar-phase (tee)	(14.56)
(deg (list 134.9633964L0 477198.8675055L0		2	;; TYPE moment -> phase	
0.0087414L0 1/69699 -1/14712000)))		3	;; Lunar phase, as an angle in degrees, at moment tee.	
360))		4	;; An angle of 0 means a new moon, 90 degrees means the	
		5	;; first quarter, 180 means a full moon, and 270 degrees	
		6	;; means the last quarter.	
		7	(let* ((phi (mod (- (lunar-longitude tee)	
(defun moon-node (c)	(14.53)	8	(solar-longitude tee))	
;; TYPE century -> angle		9	360))	
;; Moon's argument of latitude (in degrees) at moment		10	(t0 (nth-new-moon 0))	
;; given in Julian centuries c.		11	<pre>(n (round (/ (- tee t0) mean-synodic-month)))</pre>	
;; Adapted from "Astronomical Algorithms" by Jean Meeus,		12	(phi-prime (* (deg 360)	
;; Willmann-Bell, 2nd edn., 1998, p. 338.		13	<pre>(mod (/ (- tee (nth-new-moon n)))</pre>	
(mod		14	mean-synodic-month)	
(poly c		15	1))))	
(deg (list 93.2720950L0 483202.0175233L0		16	(if (> (abs (- phi phi-prime)) (deg 180)) ; close call	
-0.0036539L0 -1/3526000 1/863310000)))		17	phi-prime	
360))		18	phi)))	

(14.52)

(defun lunar-anomaly (c)

;; TYPE century -> angle

;; given in Julian centuries c.

1	(defun lunar-phase-at-or-before (phi tee)	(14.57)	3	;; Excess of lunar longitude over solar longitude at full	
2	;; TYPE (phase moment) -> moment		4	;; moon.	
3	;; Moment UT of the last time at or before tee		5	(deg 180))	
4	;; when the lunar-phase was phi degrees.				
5	(let* ((tau ; Estimate.				
6	(- tee		1	(defconstant first-quarter	(14.60)
7	(* mean-synodic-month (/ 1 (deg 360))		2	;; TYPE phase	
8	(mod (- (lunar-phase tee) phi) 360))))		3	;; Excess of lunar longitude over solar longitude at first	
9	(a (- tau 2))		4	;; quarter moon.	
10	(b (min tee (+ tau 2)))) ; At or before tee.		5	(deg 90))	
11	(invert-angular lunar-phase phi				
12	(interval-closed a b))))				
			1	(defconstant last-quarter	(14.62)
			2	;; TYPE phase	
1	(defun lunar-phase-at-or-after (phi tee)	(14.58)	3	;; Excess of lunar longitude over solar longitude at last	
2	;; TYPE (phase moment) -> moment		4	;; quarter moon.	
3	;; Moment UT of the next time at or after tee		5	(deg 270))	
4	;; when the lunar-phase is phi degrees.				
5	(let* ((tau ; Estimate.				
	(+ + + + + + + + + + + + + + + + + + +		1	(defun lunar-latitude (tee)	(14.63)
6	(+ 166				
6 7	(* mean-synodic-month (/ 1 (deg 360))		2	;; TYPE moment -> half-circle	
6 7 8	(* Lee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360))))		2 3	;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment <i>tee</i> .	
6 7 8 9	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360))</pre>		2 3 4	;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus,	
6 7 8 9 10	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2)))</pre>		2 3 4 5	;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342.	
6 7 8 9 10 11	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>		2 3 4 5 6	;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee))	
6 7 8 9 10 11 12	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>		2 3 4 5 6 7	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee))</pre>	
6 7 8 9 10 11 12	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi (interval-closed a b))))</pre>		2 3 4 5 6 7 8	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c))</pre>	
6 7 8 9 10 11 12	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>		2 3 4 5 6 7 8 9	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c))</pre>	
6 7 8 9 10 11 12	<pre>(* tee</pre>	(14.59)	2 3 4 5 6 7 8 9 10	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c)) (cap-M-prime (lunar-anomaly c))</pre>	
6 7 8 9 10 11 12 1 2	<pre>(* tee</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c)) (cap-F (moon-node c))</pre>	
6 7 8 9 10 11 12 1 2 3	<pre>(* tee</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c)) (cap-F (moon-node c)) (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0)))</pre>	
6 7 8 9 10 11 12 1 2 3 4	<pre>(* tee</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12 13	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee))</pre>	
6 7 8 9 10 11 12 1 2 3 4 5	<pre>(* tee</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12 13 14	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee))</pre>	
6 7 8 9 10 11 12 1 2 3 4 5	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12 13 14 15	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c)) (cap-F (moon-node c)) (cap-F (moon-node c)) (caps-lunar-elongation (list 0 0 0 2 2 2 2 0 2 0 2 2 2 2 2 0 4 0 0 0</pre>	
6 7 8 9 10 11 12 1 2 3 4 5	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-D (lunar-elongation c)) (cap-M (solar-anomaly c)) (cap-M-prime (lunar-anomaly c)) (cap-F (moon-node c)) (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) (args-lunar-elongation (list 0 0 0 2 2 2 2 0 2 0 2 2 2 2 2 2 2 0 4 0 0 0</pre>	
6 7 8 9 10 11 12 3 4 5	<pre>(* tee (* mean-synodic-month (/ 1 (deg 360)) (mod (- phi (lunar-phase tee)) 360)))) (a (max tee (- tau 2))) ; At or after tee. (b (+ tau 2))) (invert-angular lunar-phase phi</pre>	(14.59)	2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17	<pre>;; TYPE moment -> half-circle ;; Latitude of moon (in degrees) at moment tee. ;; Adapted from "Astronomical Algorithms" by Jean Meeus, ;; Willmann-Bell, 2nd edn., 1998, pp. 338-342. (let* ((c (julian-centuries tee)) (cap-L-prime (mean-lunar-longitude c)) (cap-U-prime (mean-lunar-longitude c)) (cap-M (solar-anomaly c)) (cap-M (solar-anomaly c)) (cap-F (moon-node c)) (cap-F (moon-node c)) (cap-E (poly c (list 1 -0.002516L0 -0.0000074L0))) (args-lunar-elongation (list 0 0 0 2 2 2 2 0 2 0 2 2 2 2 2 2 4 0 0 0</pre>	

	0 1 0 1 1 1 0 0 0 0 0 0 0 0 -1 0 0 0 0 1 1
	0 -1 -2 0 1 1 1 1 1 0 -1 1 0 -1 0 0 0 -1 -2))
(args-	lunar-anomaly
(list	0 1 1 0 -1 -1 0 2 1 2 0 -2 1 0 -1 0 -1 -1 -1
	0 0 -1 0 1 1 0 0 3 0 -1 1 -2 0 2 1 -2 3 2 -3
	-1 0 0 1 0 1 1 0 0 -2 -1 1 -2 2 -2 -1 1 1 -1
	0 0))
(args-	moon-node
(list	1 1 -1 -1 1 1 1 1 1 -1 -1 -1 1 1 1 1 1
	-1 1 3 1 1 1 -1 -1 -1 1 -1 1 -3 1 -3 -1 -1 1
	-1 1 -1 1 1 1 1 1 -1 3 -1 -1 1 -1 1 -1
	-1 -1 -1 -1 -1 1))
(sine-	coeff
(list	5128122 280602 277693 173237 55413 46271 32573
	17198 9266 8822 8216 4324 4200 -3359 2463 2211
	2065 -1870 1828 -1794 -1749 -1565 -1491 -1475
	-1410 -1344 -1335 1107 1021 833 777 671 607
	596 491 -451 439 422 421 -366 -351 331 315
	302 -283 -229 223 223 -220 -220 -185 181
	-177 176 166 -164 132 -119 115 107))
(beta	
(* (d	eg 1/1000000)
(s	igma ((v sine-coeff)
	(w args-lunar-elongation)
	(x args-solar-anomaly)
	(y args-lunar-anomaly)
	(z args-moon-node))
	(* v (expt cap-E (abs x))
	(sin-degrees
	(+ (* w cap-D)
	(* x cap-M)
	(* y cap-M-prime)
	(* z cap-F))))))))
(venus	(* (deg 175/1000000)
	(+ (sin-degrees
	(+ (deg 119.75L0) (* c (deg 131.849L0))

55	cap-F))
56	(sin-degrees
57	(+ (deg 119.75L0) (* c (deg 131.849L0))
58	(- cap-F)))))))
59	(flat-earth
60	(+ (* (deg -2235/1000000)
61	(sin-degrees cap-L-prime))
62	(* (deg 127/1000000) (sin-degrees
63	(- cap-L-prime cap-M-prime)))
64	(* (deg -115/1000000) (sin-degrees
65	(+ cap-L-prime cap-M-prime)))))
66	(extra (* (deg 382/1000000)
67	(sin-degrees
68	(+ (deg 313.45L0)
69	(* c (deg 481266.484L0))))))))
70	(+ beta venus flat-earth extra)))

1	(defun lunar-altitude (tee location)	(14.64)
2	;; TYPE (moment location) -> half-circle	
3	;; Geocentric altitude of moon at tee at location,	
4	;; as a small positive/negative angle in degrees, ignoring	
5	;; parallax and refraction. Adapted from "Astronomical	
6	;; Algorithms" by Jean Meeus, Willmann-Bell, 2nd edn.,	
7	;; 1998.	
8	(let* ((phi ; Local latitude.	
9	(latitude location))	
10	(psi ; Local longitude.	
11	(longitude location))	
12	(lambda ; Lunar longitude.	
13	(lunar-longitude tee))	
14	(beta ; Lunar latitude.	
15	(lunar-latitude tee))	
16	(alpha ; Lunar right ascension.	
17	(right-ascension tee beta lambda))	
18	(delta ; Lunar declination.	

19	(declination tee beta lambda))		23	-1 2 -1 1 -2 -1 -1 -2 0 1 4 0 -2 0 2 1 -2 -3	
20	(theta0 ; Sidereal time.		24	2 1 -1 3 -1))	
21	(sidereal-from-moment tee))		25	(args-moon-node	
22	(cap-H ; Local hour angle.		26	(list 0 0 0 0 0 2 0 0 0 0 0 0 0 -2 2 -2 0 0 0 0	
23	(mod (- theta0 (- psi) alpha) 360))		27	0 0 0 0 0 0 0 2 0 0 0 0 0 -2 2 0 2 0 0 0 0	
24	(altitude		28	0 0 -2 0 0 0 0 -2 -2 0 0 0 0 0 0 0 -2)))	
25	(arcsin-degrees (+ (* (sin-degrees phi)		29	(cosine-coeff	
26	(sin-degrees delta))		30	(list -20905355 -3699111 -2955968 -569925 48888 -314	9
27	(* (cos-degrees phi)		31	246158 -152138 -170733 -204586 -129620 108743	
28	(cos-degrees delta)		32	104755 10321 0 79661 -34782 -23210 -21636 2420	18
29	(cos-degrees cap-H))))))		33	30824 -8379 -16675 -12831 -10445 -11650 14403	
30	(mod3 altitude -180 180)))		34	-7003 0 10056 6322 -9884 5751 0 -4950 4130 0	
			35	-3958 0 3258 2616 -1897 -2117 2354 0 0 -1423	
			36	-1117 -1571 -1739 0 -4421 0 0 0 0 1165 0 0	
		(14.65)	37	8752))	
1	(defun lunar-distance (tee)	(14.65)	38	(correction	
2	;; TYPE moment -> distance		39	(sigma ((v cosine-coeff)	
3	;; Distance to moon (in meters) at moment tee.		40	(w args-lunar-elongation)	
4	;; Adapted from "Astronomical Algorithms" by Jean Meeus,		41	(x args-solar-anomaly)	
5	;; Willmann-Bell, 2nd edn., 1998, pp. 338-342.		42	(y args-lunar-anomaly)	
6	(let* ((c (julian-centuries tee))		43	(z args-moon-node))	
7	(cap-D (lunar-elongation c))		44	(* v (expt cap-E (abs x))	
8	(cap-M (solar-anomaly c))		45	(cos-degrees	
9	(cap-M-prime (lunar-anomaly c))		46	(+ (* w cap-D)	
10	(cap-F (moon-node c))		47	(* x cap-M)	
11	(cap-E (poly c (list 1 -0.002516L0 -0.0000074L0)))		48	(* y cap-M-prime)	
12	(args-lunar-elongation		49	(* z cap-F))))))))	
13	(list 0 2 2 0 0 0 2 2 2 2 0 1 0 2 0 0 4 0 4 2 2 1		50	(+ (mt 385000560) correction)))	
14	1 2 2 4 2 0 2 2 1 2 0 0 2 2 2 4 0 3 2 4 0 2				
15	2 2 4 0 4 1 2 0 1 3 4 2 0 1 2 2))				
16	(args-solar-anomaly				
17	(list 0 0 0 0 1 0 0 -1 0 -1 1 0 1 0 0 0 0 0 1 1		1	(defun lunar-parallax (tee location)	(14.66)
18	0 1 -1 0 0 0 1 0 -1 0 -2 1 2 -2 0 0 -1 0 0 1		2	;; TYPE (moment location) -> angle	
19	-1 2 2 1 -1 0 0 -1 0 1 0 1 0 0 -1 2 1 0 0))		3	;; Parallax of moon at tee at location.	
20	(args-lunar-anomaly		4	;; Adapted from "Astronomical Algorithms" by Jean Meeus,	
21	(list 1 -1 0 2 0 0 -2 -1 1 0 -1 0 1 0 1 1 -1 3 -2		5	;; Willmann-Bell, 2nd edn., 1998.	
22	-1 0 -1 0 1 2 0 -3 -2 -1 -2 1 0 2 0 -1 1 0		6	(let* ((geo (lunar-altitude tee location))	
7	(cap-Delta (lunar-distance tee))	21	(if early?		
----	--	----	--	---------	
8	(alt (/ (mt 6378140) cap-Delta))	22	(- (hr 6) offset)		
9	(arg (* alt (cos-degrees geo))))	23	(+ (hr 18) offset)))		
10	(arcsin-degrees arg)))	24	location))		
		25	bogus)))		
1	(defun topocentric-lunar-altitude (tee location) (14.67)				
2	;; TYPE (moment location) -> half-circle	1	(defun sine-offset (tee location alpha)	(14.69)	
3	;; Topocentric altitude of moon at tee at location,	2	;; TYPE (moment location half-circle) -> real		
4	;; as a small positive/negative angle in degrees,	3	;; Sine of angle between position of sun at		
5	;; ignoring refraction.	4	;; local time <i>tee</i> and		
6	(- (lunar-altitude tee location)	5	;; when its depression is alpha at location.		
7	(lunar-parallax tee location)))	6	;; Out of range when it does not occur.		
		7	(let* ((phi (latitude location))		
	Times of day are computed by the following functions:	8	(tee-prime (universal-from-local tee location))		
		9	(delta ; Declination of sun.		
1	(defun approx-moment-of-depression (tee location alpha early?) (14.68)	10	(declination tee-prime (deg 0L0)		
2	:: TYPE (moment location half-circle boolean) -> moment	11	(solar-longitude tee-prime))))		
3	:: Moment in local time near tee when depression angle	12	(+ (* (tan-degrees phi)		
4	:: of sun is alpha (negative if above horizon) at	13	(tan-degrees delta))		
5	:: location: early? is true when morning event is sought	14	(/ (sin-degrees alpha)		
6	:: and false for evening. Returns bogus if depression	15	(* (cos-degrees delta)		
7	:: angle is not reached.	16	(cos-degrees phi))))))		
8	(let* ((try (sine-offset tee location alpha))				
9	(date (fixed-from-moment tee))				
10	(alt (if (>= alpha 0)	1	(defun moment-of-depression (approx location alpha early?)	(14.70)	
11	(if early? date (1+ date))	2	;; TYPE (moment location half-circle boolean) -> moment		
12	(+ date (hr 12))))	3	;; Moment in local time near approx when depression		
13	(value (if (> (abs try) 1)	4	;; angle of sun is <i>alpha</i> (negative if above horizon) at		
14	(sine-offset alt location alpha)	5	;; location; early? is true when morning event is		
15	try)))	6	;; sought, and false for evening.		
16	(if (<= (abs value) 1) ; Event occurs	7	;; Returns bogus if depression angle is not reached.		
17	(let* ((offset (mod3 (/ (arcsin-degrees value) (deg 360))	8	(let* ((tee (approx-moment-of-depression		
18	(hr -12) (hr 12))))	9	approx location alpha early?)))		
19	(local-from-apparent	10	(if (equal tee bogus)		
20	(+ date	11	bogus		

12	(if (< (abs (- approx tee))		7	(+ date (hr 18)) location alpha evening)))	
13	(sec 30))		8	(if (equal result bogus)	
14	tee		9	bogus	
15	<pre>(moment-of-depression tee location alpha early?)))))</pre>		10	(standard-from-local result location))))	
1	(defconstant morning	(14.71)	1	(defun refraction (tee location)	(14.75)
2	;; TYPE boolean		2	;; TYPE (moment location) -> half-circle	
3	;; Signifies morning.		3	;; Refraction angle at moment tee at location.	
4	true)		4	;; The moment is not used.	
			5	(let* ((h (max (mt 0) (elevation location)))	
			6	(cap-R (mt 6.372d6)) ; Radius of Earth.	
1	(defun dawn (date location alpha)	(14.72)	7	(dip ; Depression of visible horizon.	
2	;; TYPE (fixed-date location half-circle) -> moment		8	(arccos-degrees (/ cap-R (+ cap-R h)))))	
3	;; Standard time in morning on fixed date at		9	(+ (mins 34) dip	
4	;; location when depression angle of sun is alpha.		10	(* (secs 19) (sqrt h)))))	
5	;; Returns bogus if there is no dawn on date.				
6	(let* ((result (moment-of-depression				
7	(+ date (hr 6)) location alpha morning)))				(11.50)
8	(if (equal result bogus)		1	(defun sunrise (date location)	(14.76)
9	bogus		2	;; TYPE (fixed-date location) -> moment	
10	(standard-from-local result location))))		3	;; Standard time of sunrise on fixed date at	
			4	;; location.	
			5	(let* ((alpha (+ (refraction (+ date (hr 6)) location)	
1	(defconstant evening	(14.73)	6	(mins 16))))	
2	;; TYPE boolean	(7	(dawn date location alpha)))	
3	;; Signifies evening.				
4	false)				
			1	(defun sunset (date location)	(14.77)
			2	;; TYPE (fixed-date location) -> moment	
1	(defun dusk (date location alpha)	(14.74)	3	;; Standard time of sunset on fixed date at	
2	·· TYPE (fixed-date location half-circle) -> moment	(17.77)	4	;; location.	
3	Standard time in evening on fixed date at		5	(let* ((alpha (+ (refraction (+ date (hr 18)) location)	
4	location when depression angle of sun is alpha		6	(mins 16))))	
5	Returns bogus if there is no dusk on <i>date</i>		7	(dusk date location alpha)))	
2	(let. (/regult /memory of depression				

1	(defun jewish-sabbath-ends (date location)	(14.80)	13	(if waning	
2	;; TYPE (fixed-date location) -> moment		14	(if (> offset 0)	
3	;; Standard time of end of Jewish sabbath on fixed date		15	(- tee -1 offset)	
4	;; at location (as per Berthold Cohn).		16	(- tee offset))	
5	(dusk date location (angle 7 5 0)))		17	(+ tee 1/2 offset)))	
			18	(rise (binary-search	
			19	1 (- approx (hr 6))	
1	(defun jewish-dusk (date location)	(14.81)	20	u (+ approx (hr 6))	
2	;; TYPE (fixed-date location) -> moment		21	<pre>x (> (observed-lunar-altitude x location)</pre>	
3	:: Standard time of Jewish dusk on fixed date		22	(deg 0))	
4	;; at location (as per Vilna Gaon).		23	(< (- u l) (mn 1)))))	
5	(dusk date location (angle 4 40 0)))		24	(if (< rise (1+ tee))	
	(25	(max (standard-from-universal rise location)	
			26	date) ; May be just before to midnight.	
1	(defum observed lunar altitude (tee legation)	(14.82)	27	;; Else no moonrise this day.	
1	(defuil observed=fundf=affitude (fee focation)	(14.82)	28	bogus)))	
2	;; TYPE (moment location) -> nall-circle	(
3	;; Observed altitude of upper limb of moon at tee at local,				
4	;; as a small positive/negative angle in degrees, including	3			
5	;; feffaction and elevation.		1	(defun moonset (date location)	(14.84)
0	(+ (topocentric=funar=aftitude tee focation)		2	;; TYPE (fixed-date location) -> moment	
/	(refraction tee location)		3	;; Standard time of moonset on fixed date at location.	
8	(mins 16)))		4	:: Returns bogus if there is no moonset on date.	
			4	,,	
			5	(let* ((tee ; Midnight.	
			4 5 6	<pre>(let* ((tee ; Midnight.</pre>	
1	(defun moonrise (date location)	(14.83)	4 5 6 7	(let* ((tee ; Midnight. (universal-from-standard date location)) (waxing (< (lunar-phase tee) (deg 180)))	
1 2	(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment	(14.83)	5 6 7 8	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3	(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location.	(14.83)	5 6 7 8 9	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4	(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date.	(14.83)	5 6 7 8 9 10	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4 5	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	5 6 7 8 9 10 11	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4 5 6	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	5 6 7 8 9 10 11 12	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4 5 6 7	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	4 5 6 7 8 9 10 11 12 13	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4 5 6 7 8	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	5 6 7 8 9 10 11 12 13 14	<pre>(let* ((tee ; Midnight.</pre>	
1 2 3 4 5 6 7 8 9	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	4 5 6 7 8 9 10 11 12 13 14 15	<pre>(let* ((tee ; Midnight. (universal-from-standard date location)) (waxing (< (lunar-phase tee) (deg 180))) (alt ; Altitude at midnight. (observed-lunar-altitude tee location)) (lat (latitude location)) (lat (latitude location)) (offset (/ alt (* 4 (- (deg 90) (abs lat))))) (approx ; Approximate setting time. (if waxing</pre>	
1 2 3 4 5 6 7 8 9 10	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight. (universal-from-standard date location)) (waning (> (lunar-phase tee) (deg 180))) (alt ; Altitude at midnight. (observed-lunar-altitude tee location)) (lat (latitude location))</pre>	(14.83)	5 6 7 8 9 10 11 12 13 14 15 16	<pre>(let* ((tee ; Midnight. (universal-from-standard date location)) (waxing (< (lunar-phase tee) (deg 180))) (alt ; Altitude at midnight. (observed-lunar-altitude tee location)) (lat (latitude location)) (lat (latitude location)) (offset (/ alt (* 4 (- (deg 90) (abs lat))))) (approx ; Approximate setting time. (if waxing</pre>	
1 2 3 4 5 6 7 8 9 10 11	<pre>(defun moonrise (date location) ;; TYPE (fixed-date location) -> moment ;; Standard time of moonrise on fixed date at location. ;; Returns bogus if there is no moonrise on date. (let* ((tee ; Midnight.</pre>	(14.83)	5 6 7 8 9 10 11 12 13 14 15 16 17	<pre>(let* ((tee ; Midnight.</pre>	

19	1 (- approx (hr 6))		1	(defun italian-from-local (tee_ell)	(14.88)
20	u (+ approx (hr 6))		2	;; TYPE moment -> moment	
21	x (< (observed-lunar-altitude x location) (deg ()))	3	;; Italian time corresponding to local time tee_ell.	
22	(< (- u l) (mn 1)))))		4	(let* ((date (fixed-from-moment tee_ell))	
23	(if (< set (1+ tee))		5	<pre>(z0 (local-zero-hour (1- tee_ell)))</pre>	
24	(max (standard-from-universal set location)		6	<pre>(z (local-zero-hour tee_ell)))</pre>	
25	date) ; May be just before to midnight.		7	(if (> tee_ell z) ; if after zero hour	
26	;; Else no moonset this day.		8	(+ tee_ell (- date -1 z)) ; then next day	
27	bogus)))		9	(+ tee_ell (- date z0)))))	
			1	(defun daytime-temporal-hour (date location)	(14.89)
		(14.05)	2	;; TYPE (fixed-date location) -> real	
1	(defconstant padua	(14.85)	3	;; Length of daytime temporal hour on fixed date at location.	
2	;; TYPE location		4	;; Returns bogus if there no sunrise or sunset on date.	
3	;; Location of Padua, Italy.		5	(if (or (equal (sunrise date location) bogus)	
4	(location (angle 45 24 28) (angle 11 53 9) (mt 18) (hr 1)))		6	(equal (sunset date location) bogus))	
			7	bogus	
			8	(/ (- (sunset date location)	
1	(defun local-zero-hour (tee)	(14.86)	9	(sunrise date location))	
2	:: TYPE moment -> moment	(1.000)	10	12)))	
3	:: Local time of dusk in Padua. Italy on date of moment tee.				
4	<pre>(let* ((date (fixed-from-moment tee)))</pre>		1	(defun nighttime-temporal-hour (date location)	(14.90)
5	(local-from-standard		2	;; TYPE (fixed-date location) -> real	
6	(+ (dusk date padua (angle 0 16 0)) ; Sunset.		3	;; Length of nighttime temporal hour on fixed date at locatic	n.
7	(mn 30)) ; Dusk.		4	;; Returns bogus if there no sunrise or sunset on date.	
8	padua)))		5	(if (or (equal (sunrise (1+ date) location) bogus)	
	a · · ·		6	(equal (sunset date location) bogus))	
			7	bogus	
			8	(/ (- (sunrise (1+ date) location)	
1	(defun local-from-italian (tee)	(14.87)	9	(sunset date location))	
2	;; TYPE moment -> moment		10	12)))	
3	;; Local time corresponding to Italian time tee.				
4	(let* ((date (fixed-from-moment tee))		1	(defun standard-from-sundial (tee location)	(14.91)
5	<pre>(z (local-zero-hour (1- tee))))</pre>		2	:: TYPE (moment location) -> moment	(1.1.21)
6	(- tee (- date z))))		3	:: Standard time of temporal moment tee at location	
			4	:: Returns bogus if temporal hour is undefined that day.	

5	(let* ((date (fixed-from-moment tee))	13	(+ (* (cos-degrees delta) (cos-degrees phi))	
6	(hour (* 24 (time-from-moment tee)))	14	(* (sin-degrees delta) (sin-degrees phi)))))	
7	(h (cond ((<= 6 hour 18); daytime today	15	(h ; Sun's altitude when shadow increases by	
8	(daytime-temporal-hour date location))	16	(mod3 (arctan-degrees ; double its length.	
9	((< hour 6) ; early this morning	17	(tan-degrees altitude)	
10	(nighttime-temporal-hour (1- date) location))	18	<pre>(1+ (* 2 (tan-degrees altitude))))</pre>	
11	(t ; this evening	19	-90 90)))	
12	<pre>(nighttime-temporal-hour date location)))))</pre>	20	(if (<= altitude (deg 0)) ; No shadow.	
13	(cond ((equal h bogus) bogus)	21	bogus	
14	((<= 6 hour 18); daytime today	22	(dusk date location (- h)))))	
15	(+ (sunrise date location) (* (- hour 6) h)))			
16	((< hour 6) ; early this morning			
17	(+ (sunset (1- date) location) (* (+ hour 6) h)))			
18	(t ; this evening	1	(defun alt-asr (date location)	(14.94)
19	(+ (sunset date location) (* (- hour 18) h))))))	2	;; TYPE (fixed-date location) -> moment	
		3	;; Standard time of asr on fixed date at location.	
		4	;; According to Shafi'i rule.	
1	(defun jewish-morning-end (date location) (14.92)	5	;; Returns bogus is no asr occurs.	
2	;; TYPE (fixed-date location) -> moment	6	(let* ((noon ; Time when sun nearest zenith.	
3	;; Standard time on fixed date at location of end of	7	(midday date location))	
4	;; morning according to Jewish ritual.	8	(phi (latitude location))	
5	(standard-from-sundial (+ date (hr 10)) location))	9	(delta ; Solar declination at noon.	
		10	(declination noon (deg 0) (solar-longitude noon)))	
		11	(altitude ; Solar altitude at noon.	
	(defun per (date legation) (14.02)	12	(arcsin-degrees	
2	(defull as (date location) (14.93)	13	(+ (* (cos-degrees delta) (cos-degrees phi))	
2	;; IIPE (IIXed-date location) -> moment	14	(* (sin-degrees delta) (sin-degrees phi)))))	
3	;; Standard time of asf on fixed date at location.	15	(h ; Sun's altitude when shadow increases by	
4	;; According to manari fulle.	16	(mod3 (arctan-degrees ; its length.	
5	;; Recurs bogus is no asi occurs.	17	(tan-degrees altitude)	
7	(Tet* ((Hoon ; Time when sun hearest zehith.	18	<pre>(1+ (tan-degrees altitude)))</pre>	
<i>'</i>	(midday date location))	19	-90 90)))	
8	(phi (latitude location))	20	(if (<= altitude (deg 0)) ; No shadow.	
9	(deria ; Solar declination at noon.	21	bogus	
10	(declination noon (deg v) (Solar-longitude noon)))	22	(dusk date location (- h)))))	
11	(attitude ; Solar attitude at noon.			
12	(arcsin-degrees			

	The functions for lunar visibility are:		1	(defun arc-of-vision (tee location)	(14.98)
			2	;; TYPE (moment location) -> half-circle	
1	(defun arc-of-light (tee)	(14.95)	3	;; Angular difference in altitudes of sun and moon	
2	;; TYPE moment -> half-circle		4	;; at moment tee at location.	
3	;; Angular separation of sun and moon		5	(- (lunar-altitude tee location)	
4	;; at moment <i>tee</i> .		6	(solar-altitude tee location)))	
5	(arccos-degrees				
6	(* (cos-degrees (lunar-latitude tee))		1	(defun bruin-best-view (date location)	(14.99)
7	(cos-degrees (lunar-phase tee)))))		2	;; TYPE (fixed-date location) -> moment	
			3	;; Best viewing time (UT) in the evening.	
			4	;; Yallop version, per Bruin (1977).	
1	(defun simple-best-view (date location)	(14.96)	5	(let* ((sun (sunset date location))	
2	;; TYPE (fixed-date location) -> moment		6	(moon (moonset date location))	
3	;; Best viewing time (UT) in the evening.		7	(best ; Best viewing time prior evening.	
4	;; Simple version.		8	(if (or (equal sun bogus) (equal moon bogus))	
5	(let* ((dark ; Best viewing time prior evening.		9	(1+ date) ; An arbitrary time.	
6	(dusk date location (deg 4.5L0)))		10	(+ (* 5/9 sun) (* 4/9 moon)))))	
7	(best (if (equal dark bogus)		11	(universal-from-standard best location)))	
8	(1+ date) ; An arbitrary time.				
9	dark)))		1	(defun vallon-criterion (date location)	(14,100)
10	(universal-from-standard best location)))		2	TYPE (fixed-date location) => boolean	(14.100)
			3	·· B D Vallon's criterion for possible	
			4	visibility of crescent moon on eve of date at location	
1	(defun shaukat-criterion (date location)	(14.97)	5	Not intended for high altitudes or polar regions	
2	;; TYPE (fixed-date location) -> boolean	. ,	6	(let+ (/tee · Best viewing time prior evening	
3	;; S. K. Shaukat's criterion for likely		7	(bruin-best-view (1- date) location))	
4	;; visibility of crescent moon on eve of date at location.		8	(phase (lumar-phase tee))	
5	;; Not intended for high altitudes or polar regions.		9	(can-D (lunar-semi-diameter tee location))	
6	(let* ((tee (simple-best-view (1- date) location))		10	(cap ARCL (arc-of-light tee))	
7	(phase (lunar-phase tee))		11	(cap-W (* cap-D) (-1) (cos-degrees cap-ARCL)))	
8	(h (lunar-altitude tee location))		12	(cap ARCV (arc-of-vision tee location))	
9	(cap-ARCL (arc-of-light tee)))		13	(e -0.14L0) : Crescent visible under perfect condition	ons.
10	(and (< new phase first-quarter)		14	(g1 (polv cap-W	
11	(<= (deg 10.6L0) cap-ARCL (deg 90))		15	(list 11.8371L0 -6.3226L0 0.7319L0 -0.1018)	LO))))
12	(> h (deg 4.1L0)))))		16	(and (< new phase first-quarter)	
			17	(> cap-ARCV (+ g1 e))))	
				· · · · · · · · · · · · · · · · · · ·	

1	(defun lunar-semi-diameter (tee location)	(14.101)	12	(- moon 30) ; Must go back a month.	
2	;; TYPE (moment location) -> half-circle		13	moon)))	
3	;; Topocentric lunar semi-diameter at moment tee and locatio	n.	14	(next d tau (visible-crescent d location))))	
4	(let* ((h (lunar-altitude tee location))				
5	<pre>(p (lunar-parallax tee location)))</pre>		1	(defun phasis-on-or-after (date location)	(14.105)
6	(* 0.27245L0 p (1+ (* (sin-degrees h) (sin-degrees p))))))		2	;; TYPE (fixed-date location) -> fixed-date	
			3	;; Closest fixed date on or after date on the eve	
			4	;; of which crescent moon first became visible at location.	
1	(defun lunar-diameter (tee)	(14.102)	5	(let* ((moon ; Prior new moon.	
2	;; TYPE moment -> angle		6	(fixed-from-moment	
3	;; Geocentric apparent lunar diameter of the moon (in		7	(lunar-phase-at-or-before new date)))	
4	;; degrees) at moment tee. Adapted from "Astronomical		8	(age (- date moon))	
5	;; Algorithms" by Jean Meeus, Willmann-Bell, 2nd edn.,		9	(tau ; Check if not visible yet on eve of date.	
6	;; 1998.		10	(if (or (<= 4 age)	
7	(/ (deg 1792367000/9) (lunar-distance tee)))		11	(visible-crescent (1- date) location))	
			12	(+ moon 29) ; Next new moon	
			13	date)))	
1	(defun visible-crescent (date location)	(14.103)	14	(next d tau (visible-crescent d location))))	
2	;; TYPE (fixed-date location) -> boolean				
2 3	;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon				
2 3 4	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location.</pre>			D.15 The Persian Calendar	
2 3 4 5	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another.</pre>		1	D.15 The Persian Calendar	
2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location))</pre>		1 2	D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day)	
2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location))</pre>		1 2 3	D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date	
2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location))</pre>		1 2 3 4	D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day))	
2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location)</pre>	(14.104)	1 2 3 4	D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day))	
2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date</pre>	(14.104)	1 2 3 4	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch</pre>	(15.1)
2 3 4 5 6 1 2 3	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent</pre>	(14.104)	1 2 3 4 1 2	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date</pre>	(15.1)
2 3 4 5 6 1 2 3 4	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location.</pre>	(14.104)	1 2 3 4 1 2 3	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar.</pre>	(15.1)
2 3 4 5 6 1 2 3 4 5	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19)))</pre>	(15.1)
2 3 4 5 6 1 2 3 4 5 6	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19)))</pre>	(15.1)
2 3 4 5 6 1 2 3 4 5 6 7	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4 1	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19))) (defconstant tehran</pre>	(15.1)
2 3 4 5 6 1 2 3 4 5 6 7 8	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4 1 2	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ;; TYPE (persian-year persian-month persian-day) ;; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19))) (defconstant tehran ;; TYPE location</pre>	(15.1) (15.2)
2 3 4 5 6 1 2 3 4 5 6 7 8 9	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location)) (defun phasis-on-or-before (date location) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4 1 2 3	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ; TYPE (persian-year persian-month persian-day) ; TYPE -> persian-date (list year month day)) (defconstant persian-epoch ; TYPE fixed-date ; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19))) (defconstant tehran ; TYPE location ; Location of Tehran, Iran.</pre>	(15.1) (15.2)
2 3 4 5 6 1 2 3 4 5 6 7 8 9 10	<pre>;; TYPE (fixed-date location) -> boolean ;; Criterion for possible visibility of crescent moon ;; on eve of date at location. ;; Shaukat's criterion may be replaced with another. (shaukat-criterion date location)) (defun phasis-on-or-before (date location)) (defun phasis-on-or-before (date location)) ;; TYPE (fixed-date location) -> fixed-date ;; Closest fixed date on or before date when crescent ;; moon first became visible at location. (let* ((moon ; Prior new moon.</pre>	(14.104)	1 2 3 4 1 2 3 4 1 2 3 4	<pre>D.15 The Persian Calendar (defun persian-date (year month day) ; TYPE (persian-year persian-month persian-day) ; TYPE (persian-date (list year month day)) (defconstant persian-epoch ;; TYPE fixed-date ;; Fixed date of start of the Persian calendar. (fixed-from-julian (julian-date (ce 622) march 19))) (defconstant tehran ;; TYPE location ;; Location of Tehran, Iran. (location (deg 35.68L0) (deg 51.42L0)</pre>	(15.1) (15.2)

1	(defun midday-in-tehran (date)	(15.3)	1	(defun persian-from-fixed (date)	(15.6)
2	;; TYPE fixed-date -> moment		2	;; TYPE fixed-date -> persian-date	
3	;; Universal time of true noon on fixed date in Tehran.		3	;; Astronomical Persian date (year month day)	
4	(midday date tehran))		4	;; corresponding to fixed date.	
			5	(let* ((new-year	
			6	(persian-new-year-on-or-before date))	
1	(defun persian-new-year-on-or-before (date)	(15.4)	7	(y (1+ (round (/ (- new-year persian-epoch)	
2	;; TYPE fixed-date -> fixed-date		8	<pre>mean-tropical-year))))</pre>	
3	;; Fixed date of Astronomical Persian New Year on or		9	(year (if (< 0 y)	
4	;; before fixed date.		10	У	
5	(let* ((approx ; Approximate time of equinox.		11	(1- y))); No year zero	
6	(estimate-prior-solar-longitude		12	(day-of-year (1+ (- date	
7	<pre>spring (midday-in-tehran date))))</pre>		13	(fixed-from-persian	
8	(next day (- (floor approx) 1)		14	(persian-date year 1 1)))))	
9	<pre>(<= (solar-longitude (midday-in-tehran day))</pre>		15	(month (if (<= day-of-year 186)	
10	(+ spring (deg 2))))))		16	(ceiling (/ day-of-year 31))	
			17	(ceiling (/ (- day-of-year 6) 30))))	
			18	(day ; Calculate the day by subtraction	
1	(defun fixed-from-persian (p-date)	(15.5)	19	(- date (1- (fixed-from-persian	
2	;; TYPE persian-date -> fixed-date		20	(persian-date year month 1))))))	
3	;; Fixed date of Astronomical Persian date <i>p</i> -date.		21	(persian-date year month day)))	
4	(let* ((month (standard-month p-date))				
5	(day (standard-day p-date))				
6	(year (standard-year p-date))				
7	(new-year		1	(defun arithmetic-persian-leap-year? (p-year)	(15.7)
8	(persian-new-year-on-or-before		2	;; TYPE persian-year -> boolean	
9	(+ persian-epoch 180; Fall after epoch.		3	;; True if <i>p-year</i> is a leap year on the Persian calendar.	
10	(floor		4	(let* ((y ; Years since start of 2820-year cycles	
11	(* mean-tropical-year		5	(if (< 0 p-year)	
12	(if (< 0 year)		6	(- p-year 474)	
13	(1- year)		7	(- p-year 473))); No year zero	
14	year)))))); No year zero.		8	(year ; Equivalent year in the range 4743263	
15	(+ (1- new-year) ; Days in prior years.		9	(+ (mod y 2820) 474)))	
16	(if (<= month 7) ; Days in prior months this year.		10	(< (mod (* (+ year 38)	
17	(* 31 (1- month))		11	31)	
18	(+ (* 30 (1- month)) 6))		12	128)	
19	<pre>day))) ; Days so far this month.</pre>		13	31)))	

1	(defun fixed-from-arithmetic-persian (p-date) (15.8)	10	(d1 ; Prior days not in n2820that is, days	
2	;; TYPE persian-date -> fixed-date	11	; since start of last 2820-year cycle	
3	;; Fixed date equivalent to Persian date p-date.	12	(mod d0 1029983))	
4	(let* ((day (standard-day p-date))	13	(y2820 ; Years since start of last 2820-year cycle	
5	(month (standard-month p-date))	14	(if (= d1 1029982)	
6	(p-year (standard-year p-date))	15	;; Last day of 2820-year cycle	
7	(y ; Years since start of 2820-year cycle	16	2820	
8	(if (< 0 p-year)	17	;; Otherwise use cycle of years formula	
9	(- p-year 474)	18	(quotient (+ (* 128 d1) 46878)	
10	(- p-year 473))); No year zero	19	46751)))	
11	(year ; Equivalent year in the range 4743263	20	(year ; Years since Persian epoch	
12	(+ (mod y 2820) 474)))	21	(+ 474 ; Years before start of 2820-year cycles	
13	(+ (1- persian-epoch); Days before epoch	22	(* 2820 n2820) ; Years in prior 2820-year cycles	
14	(* 1029983 ; Days in 2820-year cycles	23	y2820))); Years since start of last 2820-year	
15	; before Persian year 474	24	; cycle	
16	(quotient y 2820))	25	(if (< 0 year)	
17	(* 365 (1- year)) ; Nonleap days in prior years this	26	year	
18	; 2820-year cycle	27	(1- year)))); No year zero	
19	(quotient ; Leap days in prior years this			
20	; 2820-year cycle	1	(defun arithmetic-persian-from-fixed (date)	(15.10)
21	(- (* 31 year) 5) 128)	2	;; TYPE fixed-date -> persian-date	
22	(if (<= month 7) ; Days in prior months this year	3	;; Persian date corresponding to fixed date.	
23	(* 31 (1- month))	4	(let* ((year (arithmetic-persian-year-from-fixed date))	
24	(+ (* 30 (1- month)) 6))	5	(day-of-year (1+ (- date	
25	<pre>day))) ; Days so far this month</pre>	6	(fixed-from-arithmetic-persian	
		7	(persian-date year 1 1))))	
		8	(month (if (<= day-of-year 186)	
	(defun arithmetic pergian year from fixed (date) (150)	9	(ceiling (/ day-of-year 31))	
2	(Use and a second secon	10	(ceiling (/ (- day-of-year 6) 30))))	
2	Porsian year corresponding to the fixed date	11	(day ; Calculate the day by subtraction	
4	(lot+ ((d)	12	(- date (1- (fixed-from-arithmetic-persian	
5	, boginning in Porgian year 474	13	(persian-date year month 1))))))	
6	(_ dato (fixed_from_arithmotic_porgian	14	(persian-date year month day)))	
7	(nergian-date 475 1 1)))			
8	(n2820 · Completed prior 2820-year cycles	1	(defun nowruz (a-vear)	(15.11)
9	(motient d0 1029983))	2	·· TYPE gregorian-year -> fixed-date	(15.11)
	(Arectone do Toppion))	2	,, its gregorian year > tixed date	

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029

3	;; Fixed date of Persian New Year (Nowruz) in Gregorian
4	;; year g-year.
5	(let* ((persian-year
6	(1+ (- g-year
7	(gregorian-year-from-fixed
8	persian-epoch)))))
9	(y (if (<= persian-year 0)
10	;; No Persian year 0
11	(1- persian-year)
12	persian-year)))
13	(fixed-from-persian (persian-date y 1 1))))

	D.16 The Bahá'í Calendar	·	0,
1	(defun bahai-date (major cycle year month day)		
2	;; TYPE (bahai-major bahai-cycle bahai-year	1	(defconstant bahai-epoch
3	;; TYPE bahai-month bahai-day) -> bahai-date	2	;; TYPE fixed-date
4	(list major cycle year month day))	3	;; Fixed date of start of Baha'i calendar.
		4	(fixed-from-gregorian (gregorian-date 1844 march 21))
1	(defun bahai-major (date)		
2	;; TYPE bahai-date -> bahai-major	1	(defun fixed-from-bahai (b-date)
3	(first date))	2	;; TYPE bahai-date -> fixed-date
		3	;; Fixed date equivalent to the Baha'i date b-date.
		4	(let* ((major (bahai-major b-date))
1	(defun bahai-cycle (date)	5	(cycle (bahai-cycle b-date))
2	;; TYPE bahai-date -> bahai-cycle	6	(year (bahai-year b-date))
3	(second date))	7	(month (bahai-month b-date))
		8	(day (bahai-day b-date))
		9	(g-year; Corresponding Gregorian year.
		10	(+ (* 361 (1- major))
1	(defun bahai-year (date)	11	(* 19 (1- cycle)) year -1
2	;; TYPE bahai-date -> bahai-year	12	(gregorian-year-from-fixed bahai-epoch))))
3	(third date))	13	(+ (fixed-from-gregorian ; Prior years.
		14	(gregorian-date g-year march 20))

(defun bahai-month (date) 1 2

- ;; TYPE bahai-date -> bahai-month
- 3 (fourth date))
- (defun bahai-day (date) 1
- ;; TYPE bahai-date -> bahai-day 2
- (fifth date)) 3
 - (defconstant ayyam-i-ha (16.1)
- 2 ;; TYPE bahai-month
- ;; Signifies intercalary period of 4 or 5 days. 3
- 0) 4

1

(16.2)

(16.3)

15	(cond ((= month ayyam-i-ha) ; Intercalary period.	27	(bahai-date major cycle year
16	342) ; 18 months have elapsed.	28	ayyam-i-ha 1)))
17	((= month 19); Last month of year.	29	ayyam-i-ha) ; Intercalary period.
18	(if (gregorian-leap-year? (1+ g-year))	30	(t (1+ (quotient days 19)))))
19	347 ; Long ayyam-i-ha.	31	(day (- date -1
20	346)); Ordinary ayyam-i-ha.	32	(fixed-from-bahai
21	(t (* 19 (1- month)))); Elapsed months.	33	(bahai-date major cycle year month 1)))))
22	day))) ; Days of current month.	34	(bahai-date major cycle year month day)))

			1	(defconstant bahai-location	(16.5)
1	(defun bahai-from-fixed (date)	(16.4)	2	;; TYPE location	
2	;; TYPE fixed-date -> bahai-date		3	;; Location of Tehran for astronomical Baha'i calendar.	
3	;; Baha'i (major cycle year month day) corresponding to fixed		4	(location (deg 35.696111L0) (deg 51.423056L0)	
4	;; date.		5	$(mt \ 0) \ (hr \ (+ \ 3 \ 1/2))))$	
5	(let* ((g-year (gregorian-year-from-fixed date))				
6	(start ; 1844				
7	(gregorian-year-from-fixed bahai-epoch))		1	(defun bahai-sunset (date)	(16.6)
8	(years ; Since start of Baha'i calendar.		2	;; TYPE fixed-date -> moment	
9	(- g-year start		3	;; Universal time of sunset on fixed date	
10	(if (<= date		4	;; in Bahai-Location.	
11	(fixed-from-gregorian		5	(universal-from-standard	
12	(gregorian-date g-year march 20)))		6	(sunset date bahai-location)	
13	1 0)))		7	bahai-location))	
14	(major (1+ (quotient years 361)))				
15	(cycle (1+ (quotient (mod years 361) 19)))				
16	(year (1+ (mod years 19)))		1	(defun astro-bahai-new-year-on-or-before (date)	(16.7)
17	(days; Since start of year		2	;; TYPE fixed-date -> fixed-date	
18	(- date (fixed-from-bahai		3	;; Fixed date of astronomical Bahai New Year on or before fix	ed
19	<pre>(bahai-date major cycle year 1 1))))</pre>		4	;; date.	
20	(month		5	(let* ((approx ; Approximate time of equinox.	
21	(cond ((>= date		6	(estimate-prior-solar-longitude	
22	(fixed-from-bahai		7	<pre>spring (bahai-sunset date))))</pre>	
23	(bahai-date major cycle year 19 1)))		8	(next day (1- (floor approx))	
24	19) ; Last month of year.		9	<pre>(<= (solar-longitude (bahai-sunset day))</pre>	
25	((>= date ; Intercalary days.		10	(+ spring (deg 2))))))	
26	(fixed-from-bahai				

1	(defun fixed-from-astro-bahai (b-date)	(16.8)	4	(let* ((new-year (astro-bahai-new-year-on-or-before date))	
2	;; TYPE bahai-date -> fixed-date		5	(years (round (/ (- new-year bahai-epoch)	
3	;; Fixed date of Baha'i date b-date.		6	<pre>mean-tropical-year)))</pre>	
4	(let* ((major (bahai-major b-date))		7	(major (1+ (quotient years 361)))	
5	(cycle (bahai-cycle b-date))		8	(cycle (1+ (quotient (mod years 361) 19)))	
6	(year (bahai-year b-date))		9	(year (1+ (mod years 19)))	
7	(month (bahai-month b-date))		10	(days; Since start of year	
8	(day (bahai-day b-date))		11	(- date new-year))	
9	(years; Years from epoch		12	(month	
10	(+ (* 361 (1- major))		13	(cond	
11	(* 19 (1- cycle))		14	((>= date (fixed-from-astro-bahai	
12	year)))		15	(bahai-date major cycle year 19 1)))	
13	(cond ((= month 19); last month of year		16	; last month of year	
14	(+ (astro-bahai-new-year-on-or-before		17	19)	
15	(+ bahai-epoch		18	((>= date	
16	(floor (* mean-tropical-year		19	(fixed-from-astro-bahai	
17	(+ years 1/2)))))		20	(bahai-date major cycle year ayyam-i-ha 1)))	
18	-20 day))		21	; intercalary month	
19	((= month ayyam-i-ha)		22	ayyam-i-ha)	
20	;; intercalary month, between 18th & 19th		23	(t (1+ (quotient days 19)))))	
21	(+ (astro-bahai-new-year-on-or-before		24	(day (- date -1	
22	(+ bahai-epoch		25	(fixed-from-astro-bahai	
23	(floor (* mean-tropical-year		26	(bahai-date major cycle year month 1)))))	
24	(- years 1/2)))))		27	(bahai-date major cycle year month day)))	
25	341 day))				
26	(t (+ (astro-bahai-new-year-on-or-before				
27	(+ bahai-epoch		1	(defun bahai-new-year (g-year)	(16.10)
28	(floor (* mean-tropical-year		2	;; TYPE gregorian-year -> fixed-date	
29	(- years 1/2)))))		3	;; Fixed date of Baha'i New Year in Gregorian year g-year.	
30	(* (1- month) 19)		4	(fixed-from-gregorian	
31	day -1)))))		5	(gregorian-date g-year march 21)))	

1 (defun astro-bahai-from-fixed (date) 2

(16.9)

- ;; TYPE fixed-date -> bahai-date
- ;; Astronomical Baha'i date corresponding to fixed date. 3
- 1 (defun naw-ruz (g-year) (16.11) ;; TYPE gregorian-year -> fixed-date 2 ;; Fixed date of Baha'i New Year (Naw-Ruz) in Gregorian 3

4	;; year g-year.		3	;; Location of Paris Observatory. Longitude corresponds	
5	(astro-bahai-new-year-on-or-before		4	;; to difference of 9m 21s between Paris time zone and	
6	(gregorian-new-year (1+ g-year))))		5	;; Universal Time.	
			6	(location (angle 48 50 11) (angle 2 20 15) (mt 27) (hr 1)))	
1	(defun feast-of-ridvan (g-year)	(16.12)	1	(defun midnight-in-paris (date)	(17.2)
2	;; TYPE gregorian-year -> fixed-date		2	:: TYPE fixed-date -> moment	()
3	;; Fixed date of Feast of Ridvan in Gregorian year g-year.		3	:: Universal time of true midnight at end of fixed date	
4	(+ (naw-ruz g-year) 31))		4	;; in Paris.	
			5	(midnight (+ date 1) paris))	
1	(defun birth-of-the-bab (g-year)	(16.13)	1	(defun french-new-vear-on-or-before (date)	(17.3)
2	;; TYPE gregorian-year -> fixed-date		2	:: TYPE fixed-date -> fixed-date	(1715)
3	;; Fixed date of the Birthday of the Bab		3	;; Fixed date of French Revolutionary New Year on or	
4	;; in Gregorian year g-year.		4	;; before fixed <i>date</i> .	
5	(let* ((ny ; Beginning of Baha'i year.		5	(let* ((approx ; Approximate time of solstice.	
6	(naw-ruz g-year))		6	(estimate-prior-solar-longitude	
7	(set1 (bahai-sunset ny))		7	autumn (midnight-in-paris date))))	
8	<pre>(m1 (new-moon-at-or-after set1))</pre>		8	(next day (- (floor approx) 1)	
9	<pre>(m8 (new-moon-at-or-after (+ m1 190)))</pre>		9	(<= autumn (solar-longitude	
10	(day (fixed-from-moment m8))		10	(midnight-in-paris day))))))	
11	(set8 (bahai-sunset day)))				
12	(if (< m8 set8)		1	(defearstant fronch-on-ch	(17.4)
13	(1+ day)		2	TYPE fixed_date	(17.4)
14	(+ day 2))))		3	Fixed date of start of the French Revolutionary	
			4	·· calendar	
			5	(fixed-from-gregorian (gregorian-date 1792 september 22)))	
	D.17 The French Revolutionary Calendar		5	(linea from gregorian (gregorian date from beptember 22,,,,	
1	(defun french-date (year month day)		1	(defun fixed-from-french (f-date)	(17.5)
2	;; TYPE (french-year french-month french-day) -> french-date		2	;; TYPE french-date -> fixed-date	
3	(list year month day))		3	;; Fixed date of French Revolutionary date.	
			4	(let* ((month (standard-month f-date))	
			5	(day (standard-day f-date))	
1	(defconstant paris	(17.1)	6	(year (standard-year f-date))	
2	;; TYPE location		7	(new-year	

<pre>s (framf-mestydat-Outor-Solice 3) spring after epoch. 6 (floor (= framf-mesor) (6): spring after epoch. 6 (c) (c = framf-mesor) (1): hays in prior years 7 (c) (- (= framf-mesor) (1): hays in prior months 1 (c) (= mod f-year 4000) (1): (framf-date (1-date)) (c) (= mod f-year 4000) (1): (framf-date)) (c) (framf-date (1-date)) (c) (framf-date (1-date)) (c) (c) (framf-date (1-date)) (c) (c) (c) (c) (c) (c) (c) (c) (c) (c)</pre>	0	(french new year on an hefere		e	(and (-)(mod f y car A) 0)	
<pre>1 (not (* if end-respond resp. sping after spond. 1</pre>	0	(floor () from on on 190. Chring ofter each		6	(and (= (mod I-year 4) 0)	
<pre>10 (b) (b) (b) (b) (b) (b) (c) (b) (c) (c) (c) (c) (c) (c) (c) (c) (c) (c</pre>	10	(11001 (+ Ifench-epoch 180; Spring after epoch.		7	(not (member (mod 1-year 400) (list 100 200 300)))	
<pre>11 (trepweight); (trepwei</pre>	10	(* mean-cropical-year		1	(not (= (not 1-year 4000) 0))))	
11 (* 104 - year 1 /, bays in prior months 1 (defun fixed-from-arithmetic-french (f-date) (17.9) 1 (* 30 (1-month); bays in prior months 1; TYPE fixed-date 1; TYPE fixed-date (17.9) 1 (defun french-from-fixed (date) (17.6) (17.6) (17.6) (17.6) 1 (defun french-from-fixed (date) (17.6) (17.6) (17.6) (17.6) (17.6) (17.6) 1 (defun french-from-fixed (date) (17.6) (17.7) (17.6) (17.6) (17.7) (17	12	(1- year))))				
13 (1 = molth); bays in pind molths 2 ;; TYPE french-date -> fixed-date 1 (day)) ; Days this month 3 ;; Fixed date of nithmetic French Revolutionary 1 (defun french-from-fixed (date) (17.6) (12+* ((month (standard-day f-date)) 2 ;; TYPE fixed-date -> french-date 7 (year (standard-year f-date)) 3 ;; French Revolutionary date of fixed date. 8 (+ french-epoch -1; Days before start of calendar. 4 (let* ((new-year 9 (+ 365 (1- year)); Ordinary days in prior years. (french-new-year-on-or-before date)) 6 (year (1+ (quotient (- date new-year)))) 10 (quotient (1- year) 40) 7 mean-tropical-year))) 12 (- (quotient (1- year) 400)) 8 (month (1+ (quotient (- date new-year) 30))) 13 (quotient (1- year) 400) 10 (french-date year month day))) 15 (+ 30 (1- month); Days in prior months this year. 1 (defun french-leap-year? (f-year) (17.0) 1 (defun arithmetic-french-from-fixed (date) (17.0) 1 (defun french-leap-year? 1-> bolean 2 ;; TYPE fixed-date -> french-date (17.0) 1 (defun fre	12	(+ new-year =1 ; Days in prior years		1	(defun fixed-from-arithmetic-french (f-date)	(17.9)
i day()) j bays this month j; Fixed date of Arithmetic French Revolutionary i (defun french-from-fixed (date) (lft (let* ((month (standard-month f-date))) i (defun french-from-fixed (date) (lft (let* ((month (standard-month f-date))) j; TYPE fixed-date -> french-date (upt (month (standard-year f-date))) (year (standard-year f-date)) j; TYPE fixed-date -> french-mew-year-on-or-before date)) (let* ((month (istendard-month f-date))) (s 365 (1-year)) for dinary days in prior years. i (french-mew-year-on-or-before date)) (i (quotient (1-year)) for dinary days in prior years. i (french-new-year-on-or-before date)) (i (quotient (1-year) 400) (s 465 (1-month)); Days in prior years. i (french-date year month day))) i (quotient (1-year) 400) (s 30 (1-month)); Days in prior months this year. i (defun french-leap-year? (f-year) (lif.7) (defun arithmetic-french-from-fixed (date) (lif.10) i; TYPE french-war -> bolean ;; TYPE fixed-date -> french-date ;; Arithmetic Prench-wate year (may be off by 1). (ifrench-date (1+ f-year) 1)) (let* ((aportient (- date french-epoch -2)) (ifixed-from-arithmetic-french i; TYPE french-date f-year 1)) (french-date	15	(* 30 (1- month)); Days in prior months		2	;; TYPE french-date -> fixed-date	
 i (defun french-from-fixed (date) ii (defun french-from-fixed (date) i; TYPE fixed-date -> french-date i; TYPE fixed-date -> french-date ii (french-new-year french-eqoch ii (french-new-year french-eqoch) ii (french-date (1- quotient (1- quot	14	day))) ; Days this month		3	;; Fixed date of Arithmetic French Revolutionary	
5 (let. ((month (standard-month (standard-month (-date))) 1 (defun french-from-fixed (date) (l7.6) 2 ;; TYPE fixed-date -> french-date (l7.6) 3 ;; Prench Revolutionary date of fixed date. % 4 (let. ((new-year 9 5 (french-new-year-on-or-before date)) 10 ; Leap days in prior years. 6 (year (1+ (roud // (- new-year french-epoch) 11 (quotient (1- year) 4) 7 mean-tropical-year))) 12 (- (quotient (1- year) 40) 8 (french-date (1- date new-year) 30))) 13 (quotient (1- year) 400) 9 (day (1+ (mod (- date new-year) 30))) 13 (quotient (1- year) 4000) 10 (french-date year month day))) 15 (+ 30 (1- month)) Eays in prior months this year. 10 (defun french-leap-year?) (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 1 (defun french-leap-year?) (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TYPE french-war -> boolean 2 ;; TYPE fixed-date -> french-date ;; TYPE fixed-date -> french-date				4	;; date f-date.	
1 (defun french-from-fixed (date) (17.6) 6 (day (standard-day f-date)) 2 ;; TYPE fixed-date -> french-date 7 (year (standard-year f-date)) 3 ;; TYPE fixed-date -> french-mew-year on-or-before date)) 8 (+ french-meyo-1; Day before start of calendar. 4 (let* ((new-year 9 (+ 365 (l- year)); Ordinary days in prior years. 5 (french-new-year on-or-before date)) 10 (quotient (l- year) 4) 6 (year (l+ (round (/ (- new-year french-epoch))) 11 (quotient (l- year) 40) 7 mean-tropical-year))) 12 (- (quotient (l- year) 400) 8 (month (l+ (quotient (- date new-year) 30))) 13 (quotient (l- year) 400) 9 (day (l+ (mod (- date new-year) 30))) 14 (- (quotient (l- year) 4000) 10 (french-date new-year) 30))) 15 (+ 30 (l- month)); Days in prior months this year. 10 (defun french-leap-year? (f-year) (l7.7) 1 (defun arithmetic-french-from-fixed (date) (l7.10) 1 (defun french-leap-year? (f-year) (l7.7) 1 (defun arithmetic-french-face) (l7.10) 2 ;; TYPE french-year -> bolean <td< td=""><td></td><td></td><td></td><td>5</td><td>(let* ((month (standard-month f-date))</td><td></td></td<>				5	(let* ((month (standard-month f-date))	
2 ;; TYPE fixed-date -> french-date 7 (year (standard-year f-date))) 3 ;; French Revolutionary date of fixed date. 8 (+ french-epcoh -1; Days before start of calendar. 4 (let* ((new-year 9 (* 365 (1- year)); Ordinary days in prior years. 5 (french-new-year-on-or-before date)) 10 ; Leap days in prior years. 6 (year (1+ (round) (/ (- new-year french-epch)) 11 (quotient (1- year) 400) 7 (day (1+ (mod (- date new-year) 30))) 13 (quotient (1- year) 4000) 8 (month (1+ (quotient (- date new-year) 30))) 14 (- (quotient (1- year) 4000) 10 (french-date year month day))) 15 (* 30 (1- month)); Days in prior months this year. 10 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 1 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TYPE french-year -> bolean 2 ;; TYPE fixed-date -> french-date (17.10) 3 ;; TYPE french-year -> bolean 2 ;; TYPE fixed date. (17.10) 4 (if codi	1	(defun french-from-fixed (date)	(17.6)	6	(day (standard-day f-date))	
<pre>3 ;; French Revolutionary date of fixed date. 4 (let*(Inew-year 5 (french-new-year-on-or-before date)) 5 ((year (1+ (round (/ (- new-year french-epoch) 6 (year (1+ (round (/ (- new-year french-epoch) 7 mean-tropical-year))) 10 (day (1+ (quotient (- date new-year) 30))) 10 (french-date year month day))) 10 (french-date year month day))) 11 (defun french-leap-year? (f-year) 12 (1.7) 13 (dquotient (1- year) 400) 14 (- (quotient (1- year) 400)) 15 (+ 30 (1- month)); Days in prior months this year. 16 (day)); Days this month. 11 (defun french-leap-year? (f-year) 12 (if year is a leap year on the 13 ;; TYPE french-ware french 14 (french-date (1+ f-year) 1)) 15 (1+ (quotient (- date french-epoch -2) 17 (fixed-from-french 18 (french-date f-year 1 1)) 19 (french-date f-year 1 1)) 10 (french-date f-year -3 boolean 11 (defun arithmetic-french-leap-year? (f-year) 12 (if war is a leap year on the 13 (french-date french-leap-year? (f-year) 14 (1- quotient (- date french-epoch -2) 15 (fixed-from-french 16 (french-date f-year 1 1)) 17 (fixed-from-french 18 (french-date f-year 1 1)) 19 (defun arithmetic-french-leap-year? (f-year) 10 (french-date f-year 1 1)) 11 (lefun arithmetic-french-leap-year? (f-year) 12 (if year is a leap year on the 13 (french-date approx) 14 (1- approx) 15 (1- (approx) 15 (1- (approx) 16 (1- (approx)) 17 (fixed-from-french) 18 (french-date f-year -3 boolean 19 (french-date f-year -3 boolean 10 (french-date approx 1 1)) 11 (1- approx) 12 (if year is a leap year on the 13 (if year is a leap year on the 14 (1+ (quotient t- 14 (apoteint t- 14 (apot</pre>	2	;; TYPE fixed-date -> french-date		7	(year (standard-year f-date)))	
<pre>4 (let* ((new-year 9 (+ 365 (1- year)); Ordinary days in prior years. 5 (french-new-year-on-or-before date)) 0 ; Leap days in prior years. 6 (year (1+ (round (/ (- new-year french-epoch) 11 (quotient (1- year) 4) 7 mean-tropical-year))) 12 (- (quotient (1- year) 400) 8 (month (1+ (quotient (- date new-year) 30))) 13 (quotient (1- year) 400) 9 (day (1+ (mod (- date new-year) 30))) 14 (- (quotient (1- year) 400)) 10 (french-date year month day))) 15 (+ 30 (1- month)); Days in prior months this year. 16 day)); Days this month. 1 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TYPE french-year -> bolean 2 ;; TYPE fixed-date -> french-date 3 ;; True if f-year is a leap year on the 3 ;; Arithmetic French Revolutionary date (year month day) 4 ;; of fixed date. 5 (> (- (fixed-from-french 5 (let* (laptor x ; Approximate year (may be off by 1). 6 (french-date (1+ f-year 1 1)) 6 (let* (laptor x ; Approximate year (may be off by 1). 6 (french-date f-year 1 1))) 8 (year (if (< date 9 (fixed-from-arithmetic-french 6 (fixed-from-arithmetic-french 6 (firend-hate approx 1 1))) 1 (defun arithmetic-french-leap-year? (f-year) (17.8) 12 approx) 1 (defun arithmetic-french bolean 3 ;; TYPE fixed-face approx 1 1))) 1 (defun arithmetic-french-leap-year? (f-year) 4 (17.8) 12 approx) 2 ;; TYPE firend-hyear -> bolean 3 (month ; Calculat the month by division. 3 ;; True if f-year is a leap year on the 4 (1+ (quotient 4 (let (month 5 (leap form arithmetic-french 6 (leap form bole 6 (leap fo</pre>	3	;; French Revolutionary date of fixed date.		8	(+ french-epoch -1; Days before start of calendar.	
5 (french-new-year-on-or-before date)) 10 ; Leap days in prior years. 6 (year (1+ (roud (/ (- new-year french-epoch) 11 (quotient (1- year) 4) 7 mean-tropical-year))) 12 (- (quotient (1- year) 100)) 8 (month (1+ (quotient (- date new-year) 30))) 13 (quotient (1- year) 4000) 9 (day (1+ (mod (- date new-year) 30))) 14 (- (quotient (1- year) 4000)) 10 (french-date year month day))) 15 (* 30 (1- month)); Days in prior months this year. 10 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 1 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TTPE french-year -> boolean 2 ;; TTPE fixed-date -> french-date (17.10) 4 ;; French Revolutionary calendar. 4 ;; of fixed date. (17.10) 5 (> (- (fixed-from-french 5 (let* ((approx ; Approximate year (may be off by 1). (1 6 (french-date f-year 1 1)) 8 (year (if (< date	4	(let* ((new-year		9	(* 365 (1- year)); Ordinary days in prior years.	
6 (year (1+ (round (/ (- new-year french-epoch) mean-tropical-year))) 11 (quotient (1- year) 4) 7 mean-tropical-year))) 12 (- (quotient (1- year) 40) 9 (day (1+ (mod (- date new-year) 30))) 13 (quotient (1- year) 400) 9 (day (1+ (mod (- date new-year) 30))) 14 (- (quotient (1- year) 4000) 10 (french-date year month day))) 15 (* 30 (1- month)); Days in prior months this year. 11 (defun french-leap-year? (f-year)) (17.7) 1 (defun arithmetic-french-from-fixed (date)) (17.10) 2 ;; TYPE french-year -> boolean 2 ;; TYPE fixed-date -> french-date (17.10) 2 ;; TYPE fixed-fate -> french-date 3 ;; Arithmetic Prench Revolutionary date (year month day) 4 ;; True if f-gear is a leap year on the 3 ;; Arithmetic Prench Revolutionary date (year month day) 4 ;; of fixed date. 5 (> (- (fixed-from-french 5 (let ((approx ; Approximate year (may be off by 1). 6 (french-date (1+ f-year 1 1)) 6 (1+ (quotient (- date french-epoch -2) (fixed-from-arithmetic-french 9 365)) 9 (fixed-from-arithmetic-french) (5	(french-new-year-on-or-before date))		10	; Leap days in prior years.	
7 mean-tropical-year))) 12 (- (quotient (1- year) 100)) 8 (month (1+ (quotient (- date new-year) 30))) 13 (quotient (1- year) 400) 9 (day (1+ (mod (- date new-year) 30))) 14 (- (quotient (1- year) 400) 10 (french-date year month day))) 14 (- (quotient (1- year) 400) 10 (french-date year month day))) 14 (- (quotient (1- year) 400) 11 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 12 ;; TYPE french-year -> boolean 2 ;; TYPE fixed-date -> french-date (17.10) 14 ;; True if f-year is a leap year on the 3 ;; Arithmetic French Revolutionary date (year month day) (17.10) 14 (; french-date (1+ f-year) 1 1)) 6 (1+ (quotient (- date french-epch -2) (1- quotient (- date french-epch -2) 14 (french-date (1+ f-year) 1 1)) 6 (1+ (quotient (- date quotient -french (1460969/4000))) 15 (defun arithmetic-french-leap-year? (f-year) (17.8) 1 (defun arithmetic-french-atte approx 1 1))) (1- approx) 15 (if (data 9 (firench-date approx 1 1))) <td>6</td> <td>(year (1+ (round (/ (- new-year french-epoch)</td> <td></td> <td>11</td> <td>(quotient (1- year) 4)</td> <td></td>	6	(year (1+ (round (/ (- new-year french-epoch)		11	(quotient (1- year) 4)	
 month (1+ (quotient (- date new-year) 30))) (day (1+ (mod (- date new-year) 30))) (day (1+ (mod (- date new-year) 30)))) (french-date year month day))) (french-date year month day))) (defun french-leap-year? (f-year) (17.7) (defun arithmetic-french-from-fixed (date) (17.7) (defun arithmetic french-icap year -> boolean (17.7) (defun arithmetic french-date (17.7) (defun arithmetic french-from-fixed (date) (17.10) (17.7) (17.8) (17.8) (17.8) (17.8) (17.8) (17.9) (17.9) (17.10) (11.10) (12.10) (12.10) (12.10) (13.10) (14.10) (14.10)	7	<pre>mean-tropical-year))))</pre>		12	(- (quotient (1- year) 100))	
9 (day (1+ (mod (- date new-year) 30)))) 14 (- (quotient (1- year) 4000)) 10 (french-date year month day))) 15 (+ 30 (1- month)); Days in prior months this year. 1 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TYPE french-year -> boolean 2 ;; TYPE fixed-date -> french-date (17.10) 3 ;; True if f-year is a leap year on the 3 ;; Arithmetic French Revolutionary date (year month day) 4 ;; French Revolutionary calendar. 5 (16t* ((approx ; Approximate year (may be off by 1). 6 (french-date (1+ f-year) 1 1)) 6 (1t (quotient (- date french-epoch -2) 7 (fixed-from-french 7 1460969/4000)) 8 (year (if (< date	8	(month (1+ (quotient (- date new-year) 30)))		13	(quotient (1- year) 400)	
10 (french-date year month day))) 15 (* 30 (1- month)); Days in prior months this year. 16 day))); Days this month. 16 day)); Days this month. 1 (defun french-leap-year? (f-year) (17.7) 1 (defun arithmetic-french-from-fixed (date) (17.10) 2 ;; TYPE french-year -> boolean 2 ;; TYPE fixed-date -> french-date (17.10) 3 ;; True if f-year is a leap year on the 3 ;; Arithmetic French Revolutionary date (year month day) 4 ;; French Revolutionary calendar. 4 ;; of fixed date. 5 5 (> (- (fixed-from-french 5 (let* ((approx ; Approximate year (may be off by 1). 6 6 (french-date french-good - 2) 1460969/4000)) 1460969/4000)) 8 (year (if (< date	9	(day (1+ (mod (- date new-year) 30))))		14	(- (quotient (1- year) 4000))	
16 day)); Days this month. 1 (defun french-leap-year? (f-year) (17.7) 2 ;; TYPE french-year -> boolean 2 3 ;; Tyre if f-year is a leap year on the 2 4 ;; French Revolutionary calendar. 3 5 (> (- (fixed-from-french 3 6 (french-date (1+ f-year) 1 1)) 6 7 (fixed-from-french 5 8 (year (if (< date	10	(french-date year month day)))		15	(* 30 (1- month)); Days in prior months this year.	
1(defun french-leap-year? (f-year)(17.7)1(defun arithmetic-french-from-fixed (date)(17.10)2;; TYPE french-year -> boolean2;; TYPE fixed-date -> french-date3;; True if f-year is a leap year on the3;; Arithmetic French Revolutionary date (year month day)4;; French Revolutionary calendar.4;; of fixed date.5(> (- (fixed-from-french5(let* ((approx ; Approximate year (may be off by 1).6(french-date (1+ f-year) 1 1))6(1+ (quotient (- date french-epoch -2)7(fixed-from-french71460969/4000)))8(french-date f-year 1 1))8(year (if (< date				16	day))); Days this month.	
<pre>2 ;; TYPE french-year -> boolean 3 ;; True if f-year is a leap year on the 4 ;; French Revolutionary calendar. 5 (> (- (fixed-from-french 6 (french-date (1+ f-year) 1 1)) 7 (fixed-from-french 8 (french-date f-year 1 1))) 9 365)) 1 (defun arithmetic-french-leap-year? (f-year) 1 (defun arithmetic-french-leap-year? (f-year) 2 ;; TYPE french-year -> boolean 3 ;; True if f-year is a leap year on the 4 ;; TYPE fixed-date -> french-date 3 (percent - percent - percent</pre>	1	(defun french-leap-year? (f-year)	(17.7)	1	(defun arithmetic-french-from-fixed (date)	(17.10)
<pre>3 ;; True if f-year is a leap year on the 4 ;; French Revolutionary calendar. 5 (> (- (fixed-from-french 6 (french-date (1+ f-year) 1 1)) 6 (french-date (1+ f-year) 1 1)) 7 (fixed-from-french 8 (french-date f-year 1 1))) 9 365)) 1 (defun arithmetic-french-leap-year? (f-year) 1 (defun arithmetic-french-leap-year? (f-year) 3 (j, Arithmetic French Revolutionary date (year month day) 4 ;; of fixed date. 5 (let* ((approx ; Approximate year (may be off by 1). 6 (1+ (quotient (- date french-epoch -2) 7 (fixed-from-french 8 (year (if (< date 9 (fixed-from-arithmetic-french 10 (french-date approx 1 1))) 11 (defun arithmetic-french-leap-year? (f-year) 12 ;; TYPE french-year -> boolean 3 ;; True if f-year is a leap year on the 4 (1+ (quotient 4 (1+ (quotient))) 5 (1- approx) 6 (1+ (quotient))) 7 (1+ (quotient)) 7 (1+ (quotient)) 7 (1+ (quotient)) 7 (1+ (quotient))) 7 (1+ (quotient)) 7 (1+ (quotient))) 7 (1+ (quotient)))) 7 (1+ (quotient))) 7 (1+ (quotient)))) 7 (1+ (quotient)))) 7 (1+ (quotient)))) 7 (1+ (quotient)))) 7 (1+ (quotient))))) 7 (1+ (quotient))))) 7 (1+ (quotient)))))) 7 (1+ (quotient)))))))))))))))))))))))))))))))))))</pre>	2	;; TYPE french-year -> boolean		2	·· TYPE fixed-date -> french-date	()
<pre>4 ;; French Revolutionary calendar. 5 (> (- (fixed-from-french 6 (french-date (1+ f-year) 1 1)) 7 (fixed-from-french 8 (french-date f-year 1 1))) 8 (french-date f-year 1 1))) 9 365)) 1 (defun arithmetic-french-leap-year? (f-year) 1 (defun arithmetic-french-leap-year? (f-year) 3 ;; TYPE french-year -> boolean 3 ;; True if f-year is a leap year on the 4 ;; of fixed date. 4 ;; of fixed date. 5 (let* ((approx ; Approximate year (may be off by 1). 6 (1+ (quotient (- date french-epoch -2) 7 (1460969/4000))) 8 (year (if (< date 9 (fixed-from-arithmetic-french 10 (french-date approx 1 1))) 11 (1- approx) 13 (month ; Calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient)</pre>	3	;; True if <i>f-year</i> is a leap year on the		3	·· Arithmetic French Revolutionary date (year month day)	
<pre>5 (> (- (fixed-from-french 5 (let* ((approx ; Approximate year (may be off by 1). 6 (french-date (1+ f-year) 1 1)) 6 (let* ((approx ; Approximate year (may be off by 1). 7 (fixed-from-french 7 1460969/4000))) 8 (french-date f-year 1 1))) 8 (year (if (< date 9 365)) 9 (fixed-from-arithmetic-french 10 (french-date approx 1 1))) 11 (l- approx) 1 (defun arithmetic-french-leap-year? (f-year) (l7.8) 12 approx)) 2 ;; TYPE french-year -> boolean 13 (month ; Calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient</pre>	4	;; French Revolutionary calendar.		4	of fixed date	
<pre>6 (french-date (1+ f-year) 1 1)) 6 (1et % ((upplok ~) hppformatte french upploe fre by 1). 7 (fixed-from-french 7 1460969/4000))) 8 (french-date f-year 1 1))) 8 (year (if (< date 9 365)) 9 (fixed-from-arithmetic-french 10 (french-date approx 1 1))) 1 (defun arithmetic-french-leap-year? (f-year) (17.8) 12 approx) 1 (defun arithmetic-french-leap-year? (f-year) 13 (month ; Calculate the month by division. 3 ;; Ture if f-year is a leap year on the 14 (1+ (quotient 4 whithering Derrok D</pre>	5	(> (- (fixed-from-french		5	(let+ ((approx ·) approximate year (may be off by 1)	
7 (fixed-from-french 7 1460969/4000)) 8 (french-date f-year 1 1))) 8 (year (if (< date	6	(french-date (1+ f-year) 1 1))		6	(1et (approx , approximate year (may be orr by r).	
<pre>8 (french-date f-year 1 1))) 9 365)) 9 (french-date f-year 1 1))) 10 (french-date approx 1 1))) 11 (1- approx) 11 (defun arithmetic-french-leap-year? (f-year)) 12 ;; TYPE french-year -> boolean 13 (month ; Calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient 15) (1+ (fund frem emithematic fremeh 16) (1+ (fund frem emithematic fremeh 17) 18) (1+ (fund frem emithematic fremeh 18) (1+ (fund frem em</pre>	7	(fixed-from-french		7		
9 365)) 3 (year (If (< date	8	(french-date f-year 1 1)))		, •	(year (if (< date	
9 (Itxed=from=affine=frement 10 (fremch-date approx 1 1))) 11 (1- approx) 12 approx)) 2 ;; TYPE french-year -> boolean 3 ;; True if f-year is a leap year on the 4 (1+ (quotient 4 (1+ (quotient	9	365))		0	(year (if (< date	
1 (11 ench-date approx 1 1//) 1 (1- approx) 1 (1- approx) 2 ;; TYPE french-year -> boolean 13 3 ;; True if f-year is a leap year on the 14 4 (1+ (quotient 4 (1+ (quotient				10	(french-date approx 1 1)))	
1 (defun arithmetic-french-leap-year? (f-year) (17.8) 12 approx)) 2 ;; TYPE french-year -> boolean 13 (month ; Calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient				10	(if ench-date approx i i//)	
2 ;; TYPE french-year -> boolean 13 (month ; Calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient	1	(defun arithmetic-french-lean-wear? (f-wear)	(17.8)	12	(1- approx)	
2 ,, first french year -> bootean is (month); calculate the month by division. 3 ;; True if f-year is a leap year on the 14 (1+ (quotient))	2	TVDE fronch-woar -> booloan	(17.0)	12	(month (Calculate the month by division	
y, nue ir i gear is a reap year on the 14 (14 (quotient)	2	,, This Hendrigear -> boolean		14	(month , carculate the month by division.	
$= A_{1}A_{2} + A_{2}A_{3} + A_{3}A_{3} + A$	4	,, inde in i year is a reap year on the		14	(

(18.1)

16	(french-date year 1 1)))
17	30)))
18	(day ; Calculate the day by subtraction.
19	(1+ (- date
20	(fixed-from-arithmetic-french
21	(french-date year month 1))))))
22	(french-date year month day)))

D.18 Astronomical Lunar Calendars

1 2 3 4 5	<pre>(defun babylonian-date (year month leap day) ;; TYPE (babylonian-year babylonian-month ;; TYPE babylonian-leap babylonian-day) ;; TYPE -> babylonian-date (list year month leap day))</pre>	1 2 3 4 5	(defconstant babylon ;; TYPE location ;; Location of Babylon. (location (deg 32.4794L0) (deg 44.4328L0) (mt 26) (hr (+ 3 1/2))))	(18.2)
1 2 3	(defun babylonian-year (date) ;; TYPE babylonian-date -> babylonian-year (first date))	1 2 3 4 5	<pre>(defun babylonian-criterion (date) ;; TYPE (fixed-date location) -> boolean ;; Moonlag criterion for visibility of crescent moon on ;; eve of date in Babylon. (let* ((set (sunset (1- date) babylon))</pre>	(18.3)
1 2 3	(defun babylonian-month (date) ;; TYPE babylonian-date -> babylonian-month (second date))	6 7 8 9 10	<pre>(tee (universal-from-standard set babylon)) (phase (lunar-phase tee))) (and (< new phase first-quarter) (<= (new-moon-before tee) (- tee (hr 24))) (> (moonlag (1- date) babylon) (mn 48)))))</pre>	
1 2 3	<pre>(defun babylonian-leap (date) ;; TYPE babylonian-date -> babylonian-leap (third date)) (defun babylonian-day (date)</pre>	1 2 3 4 5	<pre>(defun babylonian-new-month-on-or-before (date) ;; TYPE fixed-date -> fixed-date ;; Fixed date of start of Babylonian month on or before ;; Babylonian date. Using lag of moonset criterion. (let* ((moon ; Prior new moon.</pre>	(18.4)
2 3	;; TYPE babylonian-date -> babylonian-day (fourth date))	6 7	(fixed-from-moment (lunar-phase-at-or-before new date)))	

1

2

3

4

5

6

7

8 9 (defun moonlag (date location)

;; TYPE (fixed-date location) -> duration

(let* ((sun (sunset date location))

(cond ((equal sun bogus) bogus)

(t (- moon sun)))))

;; Returns bogus if there is no sunset on date.

(moon (moonset date location)))

;; Time between sunset and moonset on date at location.

((equal moon bogus) (hr 24)) ; Arbitrary.

8	(age (- date moon))		15	month1))	
9	(tau ; Check if not visible yet on eve of date.		16	(midmonth ; Middle of given month.	
10	(if (and (<= age 3)		17	(+ babylonian-epoch	
11	<pre>(not (babylonian-criterion date)))</pre>		18	<pre>(round (* mean-synodic-month months)) 15)))</pre>	
12	(- moon 30) ; Must go back a month.		19	(+ (babylonian-new-month-on-or-before midmonth)	
13	moon)))		20	day -1)))	
14	(next d tau (babylonian-criterion d))))				
			1	(defun babylonian-from-fixed (date)	(18.8)
			2	;; TYPE fixed-date -> babylonian-date	
1	(defconstant babylonian-epoch	(18.5)	3	;; Babylonian date corresponding to fixed date.	
2	;; TYPE fixed-date		4	(let* ((crescent ; Most recent new month.	
3	;; Fixed date of start of the Babylonian calendar		5	(babylonian-new-month-on-or-before date))	
4	;; (Seleucid era). April 3, 311 BCE (Julian).		6	(months ; Elapsed months since epoch.	
5	(fixed-from-julian (julian-date (bce 311) april 3)))		7	(round (/ (- crescent babylonian-epoch)	
			8	<pre>mean-synodic-month)))</pre>	
			9	(year (1+ (quotient (+ (* 19 months) 5) 235)))	
1	(defun babylonian-leap-year? (b-year)	(18.6)	10	(approx ; Approximate date of new year.	
2	;; TYPE babylonian-year -> boolean		11	(+ babylonian-epoch	
3	;; True if b-year is a leap year on Babylonian calendar.		12	(round (* (quotient (+ (* (1- year) 235) 13) 19)	
4	(< (mod (+ (* 7 b-year) 13) 19) 7))		13	<pre>mean-synodic-month))))</pre>	
			14	(new-year (babylonian-new-month-on-or-before	
			15	(+ approx 15)))	
1	(defun fixed-from-babylonian (b-date)	(18.7)	16	<pre>(month1 (1+ (round (/ (- crescent new-year) 29.5L0))))</pre>	
2	;; TYPE babylonian-date -> fixed-date		17	(special (= (mod year 19) 18))	
3	;; Fixed date equivalent to Babylonian date.		18	(leap (if special (= month1 7) (= month1 13)))	
4	(let* ((month (babylonian-month b-date))		19	(month (if (or leap (and special (> month1 6)))	
5	(leap (babylonian-leap b-date))		20	(1- month1)	
6	(day (babylonian-day b-date))		21	month1))	
7	(year (babylonian-year b-date))		22	(day (- date crescent -1)))	
8	(month1 ; Elapsed months this year.		23	(babylonian-date year month leap day)))	
9	(if (or leap				
10	(and (= (mod year 19) 18)		1	(defun astronomical-easter (g-year)	(18.9)
11	(> month 6)))		2	;; TYPE gregorian-year -> fixed-date	
12	<pre>month (1- month)))</pre>		3	;; Date of (proposed) astronomical Easter in Gregorian	
13	(months ; Elapsed months since epoch.		4	;; year <i>g-year</i> .	
14	(+ (quotient (+ (* (1- year) 235) 13) 19)		5	(let* ((equinox ; Spring equinox.	

6	(season-in-gregorian spring g-year))		4	;; corresponding to fixed date.	
7	(paschal-moon ; Date of next full moon.		5	(let* ((crescent ; Most recent new moon.	
8	(floor (apparent-from-universal		6	(phasis-on-or-before date islamic-location))	
9	(lunar-phase-at-or-after full equinox)		7	(elapsed-months	
10	jerusalem)))))		8	(round (/ (- crescent islamic-epoch)	
11	;; Return the Sunday following the Paschal moon.		9	<pre>mean-synodic-month))))</pre>	
12	(kday-after sunday paschal-moon)))		10	(year (1+ (quotient elapsed-months 12)))	
			11	(month (1+ (mod elapsed-months 12)))	
			12	(day (1+ (- date crescent))))	
1	(defconstant islamic-location	(18.10)	13	(islamic-date year month day)))	
2	;; TYPE location				
3	;; Sample location for Observational Islamic calendar		1	(defun month-length (date location)	(18-13)
4	;; (Cairo, Egypt).		2	(aeran month-rengen (date location)	(10.15)
5	(location (deg 30.1L0) (deg 31.3L0) (mt 200) (hr 2)))		2	Longth of lunar month based on observability at location	
			4	, which includes date	
			5	(let+ ((moon (phasis-on-or-after (1+ date) location))	
1	(defun fixed-from-observational-islamic (i-date)	(18.11)	6	(nrew (nhasis-on-or-before date location)))	
2	;; TYPE islamic-date -> fixed-date		7	(= moon prev)))	
3	;; Fixed date equivalent to Observational Islamic date		,	(moon prev///	
4	;; i-date.				
5	(let* ((month (standard-month i-date))		1	(defun early-month? (date location)	(18.14)
6	(day (standard-day i-date))		2	;; TYPE (fixed-date location) -> boolean	
7	(year (standard-year i-date))		3	;; Fixed date in location is in a month that was forced to	
8	(midmonth ; Middle of given month.		4	;; start early.	
9	(+ islamic-epoch		5	(let* ((start (phasis-on-or-before date location))	
10	(floor (* (+ (* (1- year) 12)		6	(prev (- start 15)))	
11	month -1/2)		7	(or (>= (- date start) 30)	
12	<pre>mean-synodic-month)))))</pre>		8	(> (month-length prev location) 30)	
13	(+ (phasis-on-or-before ; First day of month.		9	(and (= (month-length prev location) 30)	
14	midmonth islamic-location)		10	(early-month? prev location)))))	
15	day -1)))				
			1	(defun alt-fixed-from-observational-islamic (i-date)	(18.15)
			2	:: TYPE islamic-date -> fixed-date	(0.000)
1	(defun observational-islamic-from-fixed (date)	(18.12)	3	;; Fixed date equivalent to Observational Islamic <i>i</i> -date.	
2	;; TYPE fixed-date -> islamic-date	. /	4	;; Months are never longer than 30 days.	
3	;; Observational Islamic date (year month day)		5	(let* ((month (standard-month i-date))	

6	(day (standard-day i-date))	3	;; Saudi visibility criterion on eve of fixed date in Mecca.	
7	(year (standard-year i-date))	4	(let* ((set (sunset (1- date) mecca))	
8	(midmonth ; Middle of given month.	5	(tee (universal-from-standard set mecca))	
9	(+ islamic-epoch	6	(phase (lunar-phase tee)))	
10	(floor (* (+ (* (1- year) 12)	7	(and (< new phase first-quarter)	
11	month -1/2)	8	(> (moonlag (1- date) mecca) 0))))	
12	<pre>mean-synodic-month))))</pre>			
13	(moon (phasis-on-or-before ; First day of month.	1	(defun saudi-new-month-on-or-before (date)	(18.18)
14	midmonth islamic-location))	2	;; TYPE fixed-date -> fixed-date	
15	(date (+ moon day -1)))	3	;; Closest fixed date on or before date when Saudi	
16	(if (early-month? midmonth islamic-location) (1- date) date)))	4	;; visibility criterion held.	
		5	(let* ((moon ; Prior new moon.	
		6	(fixed-from-moment	
1	(defun alt-observational-islamic-from-fixed (date) (18.16)	7	(lunar-phase-at-or-before new date)))	
2	;; TYPE fixed-date -> islamic-date	8	(age (- date moon))	
3	;; Observational Islamic date (year month day)	9	(tau ; Check if not visible yet on eve of date.	
4	;; corresponding to fixed date.	10	(if (and (<= age 3)	
5	;; Months are never longer than 30 days.	11	(not (saudi-criterion date)))	
6	(let* ((early (early-month? date islamic-location))	12	(- moon 30) ; Must go back a month.	
7	(long (and early	13	moon)))	
8	(> (month-length date islamic-location) 29)))	14	(next d tau (saudi-criterion d))))	
9	(date-prime			
10	(ir long (i+ date) date))	1	(defun fixed-from-saudi-islamic (s-date)	(18.19)
11	(moon ; Most recent new moon.	2	·· TYPE islamic-date -> fixed-date	(10.17)
12	(pnasis-on-or-before date-prime islamic-location))	3	Fixed date equivalent to Saudi Islamic date s-date	
13	(elapsed-months	4	(let* ((month (standard-month s-date))	
14	(round (/ (- moon islamic-epocn)	5	(day (standard-day s-date))	
15	mean-synodic-month)))	6	(vear (standard-vear s-date))	
16	(Year (1+ (quotient elapsed-months 12)))	7	(midmonth · Middle of given month	
17	(month (1+ (mod elapsed-months 12)))	8	(+ islamic-epoch	
18	(day (- date-prime moon	9	(floor (* (+ (* (1 - vear) 12))))	
19	(11 (and early (not long)) -2 -1))))	10	month $-1/2$)	
20	(ISIAMIC-date year month day)))	11	mean-synodic-month)))))	
		12	(+ (saudi-new-month-on-or-before ; First dav of month.	
1	(defun saudi-criterion (date) (18.17)	13	midmonth)	
2	;; TYPE fixed-date -> boolean	14	day -1)))	

1	(defun saudi-islamic-from-fixed (date)	(18.20)	1	(defun observational-hebrew-from-fixed (date)	(18.23)
2	;; TYPE fixed-date -> islamic-date		2	;; TYPE fixed-date -> hebrew-date	
3	;; Saudi Islamic date (year month day) corresponding to		3	;; Observational Hebrew date (year month day)	
4	;; fixed date.		4	;; corresponding to fixed date.	
5	(let* ((crescent ; Most recent new moon.		5	(let* ((crescent ; Most recent new moon.	
6	(saudi-new-month-on-or-before date))		6	(phasis-on-or-before date hebrew-location))	
7	(elapsed-months		7	(g-year (gregorian-year-from-fixed date))	
8	(round (/ (- crescent islamic-epoch)		8	(ny (observational-hebrew-first-of-nisan g-year))	
9	<pre>mean-synodic-month)))</pre>		9	(new-year (if (< date ny)	
10	(year (1+ (quotient elapsed-months 12)))		10	(observational-hebrew-first-of-nisan	
11	(month (1+ (mod elapsed-months 12)))		11	(1- g-year))	
12	(day (1+ (- date crescent))))		12	ny))	
13	(islamic-date year month day)))		13	<pre>(month (1+ (round (/ (- crescent new-year) 29.5L0))))</pre>	
			14	(year (+ (standard-year (hebrew-from-fixed new-year))	
			15	(if (>= month tishri) 1 0)))	
1	(defconstant hebrew-location	(18.21)	16	(day (- date crescent -1)))	
2	;; TYPE location		17	(hebrew-date year month day)))	
3	;; Sample location for Observational Hebrew calendar				
4	;; (Haifa, Israel).				
5	(location (deg 32.82L0) (deg 35) (mt 0) (hr 2)))				
			1	(defun fixed-from-observational-hebrew (h-date)	(18.24)
			2	;; TYPE hebrew-date -> fixed-date	
1	(defun observational-hebrew-first-of-nisan (g-year)	(18.22)	3	;; Fixed date equivalent to Observational Hebrew date.	
2	;; TYPE gregorian-year -> fixed-date		4	(let* ((month (standard-month h-date))	
3	;; Fixed date of Observational (classical)		5	(day (standard-day h-date))	
4	;; Nisan 1 occurring in Gregorian year g-year.		6	(year (standard-year h-date))	
5	(let* ((equinox ; Spring equinox.		7	(year1 (if (>= month tishri) (1- year) year))	
6	(season-in-gregorian spring g-year))		8	(start (fixed-from-hebrew	
7	(set ; Moment (UT) of sunset on day of equinox.		9	(hebrew-date year1 nisan 1)))	
8	(universal-from-standard		10	(g-year (gregorian-year-from-fixed	
9	(sunset (floor equinox) hebrew-location)		11	(+ start 60)))	
10	hebrew-location)))		12	(new-year (observational-hebrew-first-of-nisan g-year))
11	(phasis-on-or-after		13	(midmonth ; Middle of given month.	
12	(- (floor equinox) ; Day of equinox		14	(+ new-year (round (* 29.5L0 (1- month))) 15)))	
13	(if ; Spring starts before sunset.		15	(+ (phasis-on-or-before ; First day of month.	
14	(< equinox set) 14 13))		16	midmonth hebrew-location)	
15	hebrew-location)))		17	day -1)))	

1	(defun classical-passover-eve (g-year)	(18.25)	5	(let* ((month (standard-month h-date))	
2	;; TYPE gregorian-year -> fixed-date		6	(day (standard-day h-date))	
3	;; Fixed date of Classical (observational) Passover Eve		7	(year (standard-year h-date))	
4	;; (Nisan 14) occurring in Gregorian year g-year.		8	(year1 (if (>= month tishri) (1- year) year))	
5	(+ (observational-hebrew-first-of-nisan g-year) 13))		9	(start (fixed-from-hebrew	
			10	(hebrew-date year1 nisan 1)))	
			11	(g-year (gregorian-year-from-fixed	
1	(defun alt-observational-hebrew-from-fixed (date)	(18.26)	12	(+ start 60)))	
2	:: TYPE fixed-date -> hebrew-date	(/	13	(new-year (observational-hebrew-first-of-nisan g-year))
3	;; Observational Hebrew date (year month day)		14	(midmonth ; Middle of given month.	
4	:: corresponding to fixed date.		15	(+ new-year (round (* 29.5L0 (1- month))) 15))	
5	:: Months are never longer than 30 days.		16	(moon (phasis-on-or-before ; First day of month.	
6	(let* ((early (early-month? date hebrew-location))		17	midmonth hebrew-location))	
7	(long (and early (> (month-length date hebrew-location)	29)))	18	(date (+ moon day -1)))	
8	(date-prime		19	(if (early-month? midmonth hebrew-location) (1- date) date)))
9	(if long (1+ date) date))				
10	(moon ; Most recent new moon.				(10.00)
11	(phasis-on-or-before date-prime hebrew-location))		1	(derconstant samaritan-rocation	(18.28)
12	(g-year (gregorian-year-from-fixed date-prime))		2	;; TIPE location	
13	(ny (observational-hebrew-first-of-nisan g-year))		4	(location of Mt. Gerizim.	
14	(new-year (if (< date-prime ny)		+	(10Cation (deg 52.1554) (deg 55.2726) (mt 861) (mt 27))	
15	(observational-hebrew-first-of-nisan				
16	(1- g-year))		1	(defun samaritan-noon (date)	(18.29)
17	ny))		2	;; TYPE fixed-date -> moment	
18	(month (1+ (round (/ (- moon new-year) 29.5L0))))		3	;; Universal time of true noon on date at Samaritan location.	
19	(year (+ (standard-year (hebrew-from-fixed new-year))		4	(midday date samaritan-location))	
20	(if (>= month tishri) 1 0)))				
21	(day (- date-prime moon		1	(defun gemeriten new meen often (tee)	(18.20)
22	(if (and early (not long)) -2 -1))))		2	(defull samarical-new-moon-arter (tee)	(18.50)
23	(hebrew-date year month day)))		2	;; IIPE moment -> IIRed-date	
			3	;; Fixed date of first new moon after of moment tee.	
			4	;; Modern carculation.	
1	(defun alt-fixed-from-observational-hebrew (h-date)	(18.27)	6	(- (apparent-from-universal (new-moon-at-or-after too)	
2	;; TYPE hebrew-date -> fixed-date		7	((apparent from aniversal (new-moon-at-of-after tee)	
3	;; Fixed date equivalent to Observational Hebrew h-date.		, 8	(hr 12)))	
4	;; Months are never longer than 30 days.		0		

1	(defun samaritan-new-moon-at-or-before (tee)	(18.31)	3	;; Fixed date of Samaritan date <i>h-date</i> .
2	;; TYPE moment -> fixed-date		4	(let* ((month (standard-month s-date))
3	;; Fixed-date of last new moon before UT moment tee.		5	(day (standard-day s-date))
4	;; Modern calculation.		6	(year (standard-year s-date))
5	(ceiling		7	(ny (samaritan-new-year-on-or-before
6	(- (apparent-from-universal (new-moon-before tee)		8	(floor (+ samaritan-epoch 50
7	samaritan-location)		9	(* 365.25L0 (- year
8	(hr 12))))		10	(ceiling (- month 5) 8)))))))
			11	(nm (samaritan-new-moon-at-or-before
			12	(+ ny (* 29.5L0 (1- month)) 15))))
1	(defconstant samaritan-epoch	(18.32)	13	(+ nm day -1)))
2	;; TYPE fixed-date			
3	;; Fixed date of start of the Samaritan Entry Era.		1	(defun samaritan-from-fixed (date) (18.35)
4	(fixed-from-julian (julian-date (bce 1639) march 15)))		2	:: TYPE fixed-date -> hebrew-date
			3	;; Samaritan date corresponding to fixed date.
			4	(let* ((moon ; First of month
1	(defun samaritan-new-vear-on-or-before (date)	(18.33)	5	(samaritan-new-moon-at-or-before
2	;; TYPE fixed-date -> fixed-date	(6	(samaritan-noon date)))
3	;; Fixed date of Samaritan New Year on or before fixed		7	(new-year (samaritan-new-year-on-or-before moon))
4	;; date.		8	(month (1+ (round (/ (- moon new-year) 29.5L0))))
5	(let* ((g-year (gregorian-year-from-fixed date))		9	(year (+ (round (/ (- new-year samaritan-epoch) 365.25L0))
6	(dates ; All possible March 11's.		10	(ceiling (- month 5) 8)))
7	(append		11	(day (- date moon -1)))
8	(julian-in-gregorian march 11 (1- g-year))		12	(hebrew-date year month day)))
9	(julian-in-gregorian march 11 g-year)			
10	(list (1+ date)))) ; Extra to stop search.			
11	(n			D.19 The Chinese Calendar
12	(final i O		1	(defun chinese-date (cycle year month leap day)
13	(<= (samaritan-new-moon-after		2	:: TYPE (chinese-cycle chinese-year chinese-month
14	(samaritan-noon (nth i dates)))		3	:: TYPE chinese-leap chinese-day) -> chinese-date
15	date))))		4	(list cycle year month leap day))
16	(samaritan-new-moon-after (samaritan-noon (nth n dates)))))		n na ann ann ann ann ann ann ann ann an
			1	(defun chinese-cycle (date)
			2	;; TYPE chinese-date -> chinese-cycle
1	(defun fixed-from-samaritan (s-date)	(18.34)	3	(first date))
2	;; TYPE hebrew-date -> fixed-date			

1	(defun chinese-year (date)		6	(location (angle 39 55 0) (angle 116 25 0)	
2	;; TYPE chinese-date -> chinese-year		7	(mt 43.5) (hr 1397/180))	
3	(second date))		8	(location (angle 39 55 0) (angle 116 25 0)	
			9	(mt 43.5) (hr 8))))))	
1	(derun chinese-month (date)				(10.2)
2	;; TYPE Chinese-date -> Chinese-month		1	(defun chinese-solar-longitude-on-or-after (lambda tee)	(19.3)
3	(third date))		2	;; TYPE (season moment) -> moment	
			3	;; Moment (Beijing time) of the first time at or after	
			4	;; tee (Beijing time) when the solar longitude	
1	(defun chinese-leap (date)		5	;; will be <i>lambda</i> degrees.	
2	;; TYPE chinese-date -> chinese-leap		6	(let* ((sun (solar-longitude-after	
3	(fourth date))		7	Lambda	
			8	(universal-from-standard	
			9	tee	
1	(defun chinese-day (date)		10	(chinese-location tee)))))	
2	;; TYPE chinese-date -> chinese-day		11	(standard-from-universal	
3	(fifth date))		12	sun	
			13	(chinese-location sun))))	
1	(defun current-major-solar-term (date)	(19.1)			
2	;; TYPE fixed-date -> integer		1	(defun major-solar-term-on-or-after (date)	(19.4)
3	;; Last Chinese major solar term (zhongqi) before fixed		2	;; TYPE fixed-date -> moment	
4	;; date.		3	;; Moment (in Beijing) of the first Chinese major	
5	(let* ((s (solar-longitude		4	;; solar term (zhongqi) on or after fixed date. The	
6	(universal-from-standard		5	;; major terms begin when the sun's longitude is a	
7	date		6	;; multiple of 30 degrees.	
8	(chinese-location date)))))		7	(let* ((s (solar-longitude (midnight-in-china date)))	
9	(amod (+ 2 (quotient s (deg 30))) 12)))		8	(l (mod (* 30 (ceiling (/ s 30))) 360)))	
			9	(chinese-solar-longitude-on-or-after 1 date)))	
1	(defun chinese-location (tee)	(19.2)			
2	:: TYPE moment -> location	()	1	(defun current-minor-solar-term (date)	(19,5)
3	:: Location of Beijing: time zone varies with tee.		2	:: TYPE fixed-date -> integer	(1).5)
4	(let* ((vear (gregorian-vear-from-fixed (floor tee))))		3	:: Last Chinese minor solar term (jiegi) before date	
5	(if (< year 1929)		4	(let* ((s (solar-longitude	

5	(universal-from-standard		7	winter (midnight-in-china (+ date 1)))))	
6	date		8	(next day (1- (floor approx))	
7	(chinese-location date)))))		9	<pre>(< winter (solar-longitude</pre>	
8	(amod (+ 3 (quotient (- s (deg 15)) (deg 30)))		10	(midnight-in-china (1+ day)))))))	
9	12)))				
					(10.0)
			1	(derun chinese-new-moon-on-or-aiter (date)	(19.9)
1	(defun minor-solar-term-on-or-after (date)	(19.6)	2	;; TYPE fixed-date -> fixed-date	
2	;; TYPE fixed-date -> moment		3	;; Fixed date (Beijing) of first new moon on or after	
3	: Moment (in Beijing) of the first Chinese minor solar		4	;; fixed date.	
4	:: term (jiegi) on or after fixed date. The minor terms		5	(let* ((tee (new-moon-at-or-after	
5	:: begin when the sun's longitude is an odd multiple of 15		6	(midnight-in-china date))))	
6	· degrees		7	(floor	
7	(let* ((s (solar-longitude (midnight-in-china date)))		8	(standard-from-universal	
8	(1) (mod		9	tee	
0	(+ (+ 30		10	(chinese-location tee)))))	
10	(coiling				
10	$\left(\left(\left(-s,\left(\log 15\right)\right),30\right)\right)$		1	(defun chinese-new-moon-before (date)	(19.10)
12	(/ (- s (deg 15)) 50)))		2	TYPE fixed-date -> fixed-date	(1).10)
12	(deg 15))		2	. Fixed date (Reijing) of first new mean before fixed	
15			3	;; Fixed date (Berjing) of first new moon before fixed	
14	(cninese-solar-longitude-on-or-alter 1 date)))		4	;; date.	
			5	(IEC* ((CEE (NEW-MOON-DEFOTE	
			0	(midnight=in=china date))))	
1	(defun midnight-in-china (date)	(19.7)	7	(Iloor	
2	;; TYPE fixed-date -> moment		8	(standard-from-universal	
3	;; Universal time of (clock) midnight at start of fixed		9	tee	
4	;; date in China.		10	(chinese-location tee)))))	
5	(universal-from-standard date (chinese-location date)))				
			1	(defun chinese-no-major-solar-term? (date)	(19.11)
			2	;; TYPE fixed-date -> boolean	
1	(defun chinese-winter-solstice-on-or-before (date)	(19.8)	3	;; True if Chinese lunar month starting on date	
2	;; TYPE fixed-date -> fixed-date		4	:: has no major solar term.	
3	;; Fixed date, in the Chinese zone, of winter solstice		5	(= (current-major-solar-term date)	
4	;; on or before fixed date.		6	(current-major-solar-term	
5	(let* ((approx ; Approximate time of solstice.		7	(chinese-new-moon-on-or-after (+ date 1)))))	
6	(estimate-prior-solar-longitude				

1	(defun chinese-prior-leap-month? (m-prime m)	(19.12)	1	(defun chinese-new-year-on-or-before (date)	(19.14)
2	;; TYPE (fixed-date fixed-date) -> boolean		2	;; TYPE fixed-date -> fixed-date	
3	;; True if there is a Chinese leap month on or after lunar		3	;; Fixed date of Chinese New Year on or before fixed date.	
4	;; month starting on fixed day m-prime and at or before		4	(let* ((new-year (chinese-new-year-in-sui date)))	
5	;; lunar month starting at fixed date m.		5	(if (>= date new-year)	
6	(and (>= m m-prime)		6	new-year	
7	(or (chinese-no-major-solar-term? m)		7	;; Got the New Year afterthis happens if date is	
8	(chinese-prior-leap-month?		8	;; after the solstice but before the new year.	
9	m-prime		9	;; So, go back half a year.	
10	(chinese-new-moon-before m)))))		10	(chinese-new-year-in-sui (- date 180)))))	
1	(defun chinese-new-year-in-sui (date)	(19.13)			
2	;; TYPE fixed-date -> fixed-date		1	(defconstant chinese-epoch	(19.15)
3	;; Fixed date of Chinese New Year in sui (period from		2	:: TYPE fixed-date	(,
4	;; solstice to solstice) containing date.		3	;; Fixed date of start of the Chinese calendar.	
5	(let* ((s1; prior solstice		4	(fixed-from-gregorian (gregorian-date -2636 february 15)))	
6	(chinese-winter-solstice-on-or-before date))				
7	(s2; following solstice				
8	(chinese-winter-solstice-on-or-before				
9	(+ s1 370)))		1	(defun chinese-from-fixed (date)	(19.16)
10	(m12 ; month after 11th montheither 12 or leap 11		2	;; TYPE fixed-date -> chinese-date	
11	(chinese-new-moon-on-or-after (1+ s1)))		3	;; Chinese date (cycle year month leap day) of fixed date.	
12	(m13 ; month after m12either 12 (or leap 12) or 1 $$		4	(let* ((s1; Prior solstice	
13	(chinese-new-moon-on-or-after (1+ m12)))		5	(chinese-winter-solstice-on-or-before date))	
14	(next-m11 ; next 11th month		6	(s2; Following solstice	
15	(chinese-new-moon-before (1+ s2))))		7	(chinese-winter-solstice-on-or-before (+ s1 370)))	
16	(if ; Either m12 or m13 is a leap month if there are		8	(m12 ; month after last 11th month	
17	; 13 new moons (12 full lunar months) and		9	(chinese-new-moon-on-or-after (1+ s1)))	
18	; either m12 or m13 has no major solar term		10	(next-m11; next 11th month	
19	(and (= (round (/ (- next-m11 m12)		11	(chinese-new-moon-before (1+ s2)))	
20	<pre>mean-synodic-month))</pre>		12	(m ; start of month containing date	
21	12)		13	(chinese-new-moon-before (1+ date)))	
22	(or (chinese-no-major-solar-term? m12)		14	(leap-year; if there are 13 new moons (12 full	
23	<pre>(chinese-no-major-solar-term? m13)))</pre>		15	; lunar months)	
24	(chinese-new-moon-on-or-after (1+ m13))		16	(= (round (/ (- next-m11 m12)	
25	m13)))		17	<pre>mean-synodic-month))</pre>	

18	12))	6	(month (chinese-month c-date))
19	(month ; month number	7	(leap (chinese-leap c-date))
20	(amod	8	(day (chinese-day c-date))
21	(-	9	(mid-year ; Middle of the Chinese year
22	;; ordinal position of month in year	10	(floor
23	(round (/ (- m m12) mean-synodic-month))	11	(+ chinese-epoch
24	;; minus 1 during or after a leap month	12	(* (+ (* (1- cycle) 60); years in prior cycles
25	(if (and leap-year	13	(1- year) ; prior years this cycle
26	(chinese-prior-leap-month? m12 m))	14	1/2) ; half a year
27	1	15	<pre>mean-tropical-year))))</pre>
28	0))	16	(new-year (chinese-new-year-on-or-before mid-year))
29	12))	17	(p ; new moon before datea month too early if
30	(leap-month ; it's a leap month if	18	; there was prior leap month that year
31	(and	19	(chinese-new-moon-on-or-after
32	leap-year;there are 13 months	20	(+ new-year (* (1- month) 29))))
33	(chinese-no-major-solar-term?	21	<pre>(d (chinese-from-fixed p))</pre>
34	m) ; no major solar term	22	(prior-new-moon
35	(not (chinese-prior-leap-month? ; and no prior leap	23	(if ; If the months match
36	; month	24	(and (= month (chinese-month d))
37	<pre>m12 (chinese-new-moon-before m)))))</pre>	25	(equal leap (chinese-leap d)))
38	(elapsed-years ; Approximate since the epoch	26	p;that's the right month
39	(floor (+ 1.5L0 ; 18 months (because of truncation)	27	;; otherwise, there was a prior leap month that
40	(- (/ month 12)); after at start of year	28	;; year, so we want the next month
41	(/ (- date chinese-epoch)	29	(chinese-new-moon-on-or-after (1+ p)))))
42	<pre>mean-tropical-year))))</pre>	30	(+ prior-new-moon day -1)))
43	(cycle (1+ (quotient (1- elapsed-years) 60)))		
44	(year (amod elapsed-years 60))	1	(defun chinese-name (stem branch)
45	(day (1+ (- date m))))	2	:: TYPE (chinese-stem chinese-branch) -> chinese-name
46	(chinese-date cycle year month leap-month day)))	3	;; Combination is impossible if stem and branch
		4	;; are not the equal mod 2.
		5	(list stem branch))
1	(defun fixed-from-chinese (c-date) (19.17)		
2	;; TYPE chinese-date -> fixed-date	1	(defun chinese-stom (name)
3	;; Fixed date of Chinese date <i>c</i> -date.	1	(uerun chinese-stem (Hame)
4	(let* ((cycle (chinese-cycle c-date))	2	(first name))
5	(year (chinese-year c-date))	3	(IIISC Hame))

1	(defun chinese-branch (name)		1	(defun chinese-month-name (month year)	(19.22)
2	:: TYPE chinese-name -> chinese-branch		2	:: TYPE (chinese-month chinese-year) -> chinese-name	()
3	(second name))		3	: Sexagesimal name for month month of Chinese year	
1 2 3 4 5	<pre>(defun chinese-sexagesimal-name (n) ;; TYPE integer -> chinese-name ;; The n-th name of the Chinese sexagesimal cycle. (chinese-name (amod n 10)</pre>	(19.18)	4 5 6 7 8	<pre>;; gear. (let* ((elapsed-months (+ (* 12 (1- year))</pre>	
1	(defun chinese-name-difference (c-name1 c-name2)	(19.19)			
2 3 4 5 6 7 8	<pre>(defun chinese-hame-drifterence (c-hamer c-hamer) ;; TYPE (chinese-name chinese-name) -> nonnegative-integer ;; Number of names from Chinese name c-name1 to the ;; next occurrence of Chinese name c-name2. (let* ((stem1 (chinese-stem c-name1)) (stem2 (chinese-stem c-name1)) (branch1 (chinese-branch c-name1)) (branch2 (chinese-branch c-name2))</pre>	(19.19)	1 2 3 4	(defconstant chinese-day-name-epoch ;; TYPE integer ;; RD date of a start of Chinese sexagesimal day cycle. (rd 45))	(19.23)
9 10 11 12 13 14	<pre>(stem-difference (- stem2 stem1)) (branch-difference (- branch2 branch1))) (amod (+ stem-difference</pre>		1 2 3 4 5	<pre>(defun chinese-day-name (date) ;; TYPE fixed-date -> chinese-name ;; Chinese sexagesimal name for date. (chinese-sexagesimal-name (- date chinese-day-name-epoch)))</pre>	(19.24)
1 2 3 4	(defun chinese-year-name (year) ;; TYPE chinese-year -> chinese-name ;; Sexagesimal name for Chinese <i>gear</i> of any cycle. (chinese-sexagesimal-name year))	(19.20)	1 2 3	(defun chinese-day-name-on-or-before (name date) ;; TYPE (chinese-name fixed-date) -> fixed-date ;; Fixed date of latest date on or before fixed <i>date</i>	(19.25)
1 2 3 4	<pre>(defconstant chinese-month-name-epoch ;; TYPE integer ;; Elapsed months at start of Chinese sexagesimal month :: cvcle.</pre>	(19.21)	4 5 6 7	<pre>;; that has Chinese name. (mod3 (chinese-name-difference (chinese-day-name 0) name) date (- date 60)))</pre>	
5	57)				

1	(defun chinese-new-year (g-year)	(19.26)	6	(let* ((today (chinese-from-fixed date)))	
2	;; TYPE gregorian-year -> fixed-date		7	(if (>= date (fixed-from-chinese birthdate))	
3	;; Fixed date of Chinese New Year in Gregorian year g-year.		8	(+ (* 60 (- (chinese-cycle today)	
4	(chinese-new-year-on-or-before		9	(chinese-cycle birthdate)))	
5	(fixed-from-gregorian		10	(- (chinese-year today)	
6	(gregorian-date g-year july 1))))		11	(chinese-year birthdate))	
			12	1)	
			13	bogus)))	
1	(defun dragon-festival (g-year)	(19.27)			
2	;; TYPE gregorian-year -> fixed-date			(defeenstant double bright	(10.20)
3	;; Fixed date of the Dragon Festival occurring in		1	(derconstant double-bright	(19.50)
4	;; Gregorian year g-year.		2	;; TYPE augury	
5	(let* ((elapsed-years		3	;; Lichun occurs twice (double-nappiness).	
6	(1+ (- g-year		4	3)	
7	(gregorian-year-from-fixed				
8	chinese-epoch))))		1	(defconstant bright	(19.31)
9	(cycle (1+ (quotient (1- elapsed-years) 60)))		2	;; TYPE augury	
10	(year (amod elapsed-years 60)))		3	:: Lichun occurs once at the start.	
11	(fixed-from-chinese (chinese-date cycle year 5 false 5))))		4	2)	
1	(defun qing-ming (g-year)	(19.28)	1	(defconstant blind	(19.32)
2	;; TYPE gregorian-year -> fixed-date		2	;; TYPE augury	
3	;; Fixed date of Qingming occurring in Gregorian year		3	;; Lichun occurs once at the end.	
4	;; g-year.		4	1)	
5	(floor				
6	(minor-solar-term-on-or-after				(10.22)
7	(fixed-from-gregorian		1	(defconstant widow	(19.33)
8	(gregorian-date g-year march 30)))))		2	;; TYPE augury	
			3	;; Lichun does not occur (double-blind year).	
			4	0)	
1	(defun chinese-age (birthdate date)	(19.29)			
2	;; TYPE (chinese-date fixed-date) -> nonnegative-integer		1	(defun chinese-year-marriage-augury (cycle year)	(19.34)
3	;; Age at fixed date, given Chinese birthdate,		2	;; TYPE (chinese-cycle chinese-year) -> augury	
4	;; according to the Chinese custom. Returns bogus if		3	;; The marriage augury type of Chinese year in cycle.	
5	;; date is before birthdate.		4	(let* ((new-year (fixed-from-chinese	

5	(chinese-date cycle year 1 false 1)))	:	(mt 24) (hr (+ 9 143/450)))	
6	(c (if (= year 60); next year's cycle		; Longitude 135 time zone	
7	(1+ cycle)	10) (location (deg 35) (deg 135) (mt 0) (hr 9)))))	
8	cycle))			
9	(y (if (= year 60); next year's number			
10	1		(defun korean-location (tee)	19.36)
11	(1+ year)))	-	2 ;; TYPE moment -> location	
12	(next-new-year (fixed-from-chinese	:	;; Location for Korean calendar; varies with tee.	
13	(chinese-date c y 1 false 1)))		4 ;; Seoul city hall at a varying time zone.	
14	(first-minor-term	:	5 (let* ((z (cond	
15	(current-minor-solar-term new-year))		5 ((< tee	
16	(next-first-minor-term		/ (fixed-from-gregorian	
17	(current-minor-solar-term next-new-year)))	:	(gregorian-date 1908 april 1)))	
18	(cond		;; local mean time for longitude 126 deg 58 min	
19	((and	10	3809/450)	
20	(= first-minor-term 1) ; no lichun at start	1	l ((< tee	
21	(= next-first-minor-term 12)) ;or at end	1:	2 (fixed-from-gregorian	
22	widow)	1	(gregorian-date 1912 january 1)))	
23	((and	14	4 8.5)	
24	(= first-minor-term 1) ; no lichun at start	1:	5 ((< tee	
25	(/= next-first-minor-term 12));only at end	10	5 (fixed-from-gregorian	
26	blind)	1	(gregorian-date 1954 march 21)))	
27	((and	1	3 9)	
28	(/= first-minor-term 1) ; lichun at start	1) ((< tee	
29	(= next-first-minor-term 12)) ; not at end	20) (fixed-from-gregorian	
30	bright)	2	(gregorian-date 1961 august 10)))	
31	(t double-bright)))) ; lichun at start and end	2:	2 8.5)	
		2.	3 (t 9))))	
		24	(location (angle 37 34 0) (angle 126 58 0)	
		2:	5 (mt 0) (hr z))))	
1	(defun japanese-location (tee)	(19.35)		
2	;; TYPE moment -> location			
3	;; Location for Japanese calendar; varies with tee.		(defun korean-year (cycle year)	19.37)
4	(let* ((year (gregorian-year-from-fixed (floor tee))))	:	2 ;; TYPE (chinese-cycle chinese-year) -> integer	
5	(if (< year 1888)	:	;; Equivalent Korean year to Chinese cycle and year	
6	;; Tokyo (139 deg 46 min east) local time		4 (+ (* 60 cycle) year -364))	
7	(location (deg 35.7L0) (angle 139 46 0)			

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029

(20.5)

1	(defun vietnamese-location (tee)	(19.38)	1	(defun hindu-sine-table (entry)	(20.4)
2	;; TYPE moment -> location		2	;; TYPE integer -> rational-amplitude	
3	;; Location for Vietnamese calendar is Hanoi; varies with		3	;; This simulates the Hindu sine table.	
4	;; tee. Time zone has changed over the years.		4	;; entry is an angle given as a multiplier of 225'.	
5	(let* ((z (if (< tee		5	(let* ((exact (* 3438 (sin-degrees	
6	(gregorian-new-year 1968))		6	(* entry (angle 0 225 0)))))	
7	8		7	(error (* 0.215L0 (sign exact)	
8	7)))		8	(sign (- (abs exact) 1716))))	
9	(location (angle 21 2 0) (angle 105 51 0)		9	(/ (round (+ exact error)) 3438)))	
10	(mt 12) (hr z))))				

1

2

3

D.20 The Modern Hindu Calendars

Common Lisp supplies arithmetic with arbitrary rational numbers, an ing the Hindu calendars. With other languages, 64-bit arithmetic is re

1	(defconstant hindu-sidereal-year	(20.1)
2	;; TYPE rational	
3	;; Mean length of Hindu sidereal year.	
4	(+ 365 279457/1080000))	

nd we take advantage of this for implement-	4	(let* ((entry
equired for many of the calculations.	5		(/ the
	6		(fract
(20.1)) 7	(+ (*	fracti
	8		(hindu
	9	(*	(- 1 f

```
theta (angle 0 225 0))); Interpolate in table.
                 action (mod entry 1)))
                 action
                 indu-sine-table (ceiling entry)))
            (* (- 1 fraction)
10
```

;; TYPE rational-angle -> rational-amplitude

;; Linear interpolation for theta in Hindu table.

```
(hindu-sine-table (floor entry))))))
```

(defun hindu-sine (theta)

```
(defun hindu-arcsin (amp)
                                                                                                                                                         (20.6)
    (defconstant hindu-sidereal-month
                                                                        (20.2)
                                                                                 1
1
                                                                                 2
                                                                                       ;; TYPE rational-amplitude -> rational-angle
2
      :: TYPE rational
                                                                                 3
                                                                                       ;; Inverse of Hindu sine function of amp.
3
      ;; Mean length of Hindu sidereal month.
                                                                                 4
                                                                                        (if (< amp 0) (- (hindu-arcsin (- amp)))
      (+ 27 \ 4644439/14438334))
4
                                                                                 5
                                                                                         (let* ((pos (next k 0 (<= amp (hindu-sine-table k))))</pre>
                                                                                                 (below ; Lower value in table.
                                                                                 6
                                                                                 7
                                                                                                  (hindu-sine-table (1- pos))))
    (defconstant hindu-synodic-month
                                                                        (20.3)
                                                                                            (* (angle 0 225 0)
1
                                                                                 8
                                                                                               (+ pos -1 ; Interpolate.
2
      ;; TYPE rational
                                                                                 9
3
      ;; Mean time from new moon to new moon.
                                                                                                  (/ (- amp below)
                                                                                10
      (+ 29 7087771/13358334))
4
                                                                                11
                                                                                                     (- (hindu-sine-table pos) below)))))))
```

1	(defun hindu-mean-position (tee period)	(20.7)	12	(hindu-sine (hindu-mean-position tee anomalistic)))	
2	;; TYPE (rational-moment rational) -> rational-angle		13	(contraction (* (abs offset) change size))	
3	;; Position in degrees at moment tee in uniform circular		14	(equation ; Equation of center	
4	;; orbit of period days.		15	(hindu-arcsin (* offset (- size contraction)))))	
5	(* (deg 360) (mod (/ (- tee hindu-creation) period) 1)))		16	(mod (- lambda equation) 360)))	
1	(defconstant hindu-creation	(20.8)	1	(defun hindu-solar-longitude (tee)	(20.12)
2	:: TYPE fixed-date	(,	2	;; TYPE rational-moment -> rational-angle	
3	;; Fixed date of Hindu creation.		3	;; Solar longitude at moment <i>tee</i> .	
4	(- hindu-epoch (* 1955880000 hindu-sidereal-vear)))		4	(hindu-true-position tee hindu-sidereal-year	
			5	14/360 hindu-anomalistic-year 1/42))	
1	(defconstant hindu-anomalistic-year	(20.9)			
2	;; TYPE rational		1	(defun hindu-zodiac (tee)	(20.13)
3	;; Time from aphelion to aphelion.		2	;; TYPE rational-moment -> hindu-solar-month	
4	(/ 1577917828000 (- 4320000000 387)))		3	;; Zodiacal sign of the sun, as integer in range 112,	
			4	;; at moment <i>tee</i> .	
			5	<pre>(1+ (quotient (hindu-solar-longitude tee) (deg 30))))</pre>	
1	(defconstant hindu-anomalistic-month	(20.10)			
2	;; TYPE rational				
3	;; Time from apogee to apogee, with bija correction.		1	(defun hindu-lunar-longitude (tee)	(20.14)
4	(/ 1577917828 (- 57753336 488199)))		2	;; TYPE rational-moment -> rational-angle	
			3	;; Lunar longitude at moment tee.	
			4	(hindu-true-position tee hindu-sidereal-month	
1	(defun hindu-true-position (tee period size anomalistic change)	(20.11)	5	32/360 hindu-anomalistic-month 1/96))	
2	;; TYPE (rational-moment rational rational rational				
3	;; TYPE rational) -> rational-angle				
4	;; Longitudinal position at moment tee. period is		1	(defun hindu-lunar-phase (tee)	(20.15)
5	;; period of mean motion in days. size is ratio of		2	;; TYPE rational-moment -> rational-angle	
6	;; radii of epicycle and deferent. anomalistic is the		3	;; Longitudinal distance between the sun and moon	
7	;; period of retrograde revolution about epicycle.		4	;; at moment <i>tee</i> .	
8	;; change is maximum decrease in epicycle size.		5	(mod (- (hindu-lunar-longitude tee)	
9	(let* ((lambda ; Position of epicycle center		6	(hindu-solar-longitude tee))	
10	(hindu-mean-position tee period))		7	360))	
11	(offset ; Sine of anomaly				

1	(defun hindu-lunar-day-from-moment (tee)	(20.16)	1	(defconstant hindu-solar-era	(20.19)
2	;; TYPE rational-moment -> hindu-lunar-day		2	;; TYPE standard-year	
3	;; Phase of moon (tithi) at moment tee, as an integer in		3	;; Years from Kali Yuga until Saka era.	
4	;; the range 130.		4	3179)	
5	(1+ (quotient (hindu-lunar-phase tee) (deg 12))))				
			1	(defun hindu-solar-from-fixed (date)	(20.20)
1	(defun hindu-new-moon-before (tee)	(20.17)	2	;; TYPE fixed-date -> hindu-solar-date	
2	;; TYPE rational-moment -> rational-moment		3	;; Hindu (Orissa) solar date equivalent to fixed date.	
3	;; Approximate moment of last new moon preceding moment		4	(let* ((critical ; Sunrise on Hindu date.	
4	;; tee, close enough to determine zodiacal sign.		5	(hindu-sunrise (1+ date)))	
5	(let* ((varepsilon (expt 2 -1000)) ; Safety margin.		6	(month (hindu-zodiac critical))	
6	(tau ; Can be off by almost a day.		7	(year (- (hindu-calendar-year critical)	
7	(- tee (* (/ 1 (deg 360)) (hindu-lunar-phase tee)		8	hindu-solar-era))	
8	hindu-synodic-month))))		9	(approx ; 3 days before start of mean month.	
9	(binary-search ; Search for phase start.		10	(- date 3	
10	l (1- tau)		11	<pre>(mod (floor (hindu-solar-longitude critical))</pre>	
11	u (min tee (1+ tau))		12	(deg 30))))	
12	x (< (hindu-lunar-phase x) (deg 180))		13	(start ; Search forward for beginning	
13	(or (= (hindu-zodiac l) (hindu-zodiac u))		14	(next i approx ; of month.	
14	<pre>(< (- u l) varepsilon)))))</pre>		15	(= (hindu-zodiac (hindu-sunrise (1+ i)))	
			16	month)))	
			17	(day (- date start -1)))	
1	(defun hindu-solar-date (year month day)		18	(hindu-solar-date year month day)))	
2	;; TYPE (hindu-solar-year hindu-solar-month hindu-solar-day)				
3	;; TYPE -> hindu-solar-date				
4	(list year month day))		1	(defun fixed-from-hindu-solar (s-date)	(20.21)
			2	;; TYPE hindu-solar-date -> fixed-date	
			3	;; Fixed date corresponding to Hindu solar date s-date	
1	(defun hindu-calendar-year (tee)	(20.18)	4	;; (Saka era; Orissa rule.)	
2	;; TYPE rational-moment -> hindu-solar-year		5	(let* ((month (standard-month s-date))	
3	;; Determine solar year at given moment tee.		6	(day (standard-day s-date))	
4	(round (- (/ (- tee hindu-epoch)		7	(year (standard-year s-date))	
5	hindu-sidereal-year)		8	(start ; Approximate start of month	
6	(/ (hindu-solar-longitude tee)		9	; by adding days	
7	(deg 360)))))		10	(+ (floor (* (+ year hindu-solar-era	

11	(/ (1- month) 12)) ; in months	1	(defun hindu-lunar-year (date)
12	hindu-sidereal-year)) ; and years	2	;; TYPE hindu-lunar-date -> hindu-lunar-year
13	hindu-epoch))) ; and days before RD 0.	3	(first date))
14	;; Search forward to correct month		
15	(+ day -1		
16	(next d (- start 3)	1	(defconstant hindu-lunar-era
17	(= (hindu-zodiac (hindu-sunrise (1+ d)))	2	;; TYPE standard-year
18	month)))))	3	;; Years from Kali Yuga until Vikrama era.
		4	3044)
1	(defun hindu-lunar-date (year month leap-month day leap-day)		
2	;; TYPE (hindu-lunar-year hindu-lunar-month	1	(defun hindu-lunar-from-fived (date)
3	;; TYPE hindu-lunar-leap-month hindu-lunar-day	2	·· TVPE fixed_date -> hindu_lupar_date
4	;; TYPE hindu-lunar-leap-day) -> hindu-lunar-date	3	·· Hindu lunar date new-moon scheme
5	(list year month leap-month day leap-day))	4	:: equivalent to fixed date.
		5	(let* ((critical (hindu-sunrise date)) ; Sunrise that day.
		6	(day (hindu-lunar-day-from-moment
1	(defun hindu-lunar-month (date)	7	critical)); Day of month.
2	;; TYPE hindu-lunar-date -> hindu-lunar-month	8	(leap-day ; If previous day the same.
3	(second date))	9	(= day (hindu-lunar-day-from-moment
		10	(hindu-sunrise (- date 1)))))
		11	(last-new-moon
1	(defun hindu-lunar-leap-month (date)	12	(hindu-new-moon-before critical))
2	;; TYPE hindu-lunar-date -> hindu-lunar-leap-month	13	(next-new-moon
3	(third date))	14	(hindu-new-moon-before
		15	(+ (floor last-new-moon) 35)))
		16	(solar-month ; Solar month name.
1	(defun hindu-lunar-day (date)	17	(hindu-zodiac last-new-moon))
2	;; TYPE hindu-lunar-date -> hindu-lunar-day	18	(leap-month ; If begins and ends in same sign.
3	(fourth date))	19	(= solar-month (hindu-zodiac next-new-moon)))
		20	(month ; Month of lunar year.
		21	(amod (1+ solar-month) 12))
1	(defun hindu-lunar-leap-day (date)	22	(year ; Solar year at end of month.
2	;; TYPE hindu-lunar-date -> hindu-lunar-leap-day	23	(- (hindu-calendar-year
3	(fifth date))	24	(if (<= month 2) ; <i>date</i> might precede solar
		25	; new year.

(20.22)

(20.23)

26	(+ date 180)		31	(/= (hindu-lunar-month mid) month)	
27	date))		32	(and (hindu-lunar-leap-month mid)	
28	hindu-lunar-era)))		33	<pre>(not leap-month))))</pre>	
29	(hindu-lunar-date year month leap-month day leap-day)))		34	(mod3 k -15 15))	
			35	(t ; In preceding month.	
			36	(mod3 k 15 45)))))	
		(20.24)	37	(tau ; Refined estimate.	
1	(defun fixed-from-hindu-lunar (l-date)	(20.24)	38	(- est (mod3 (- (hindu-lunar-day-from-moment	
2	;; TYPE hindu-lunar-date -> fixed-date		39	(+ est (hr 6)))	
3	;; Fixed date corresponding to Hindu lunar date 1-date.		40	day)	
4	(let* ((year (hindu-lunar-year l-date))		41	-15 15)))	
5	(month (hindu-lunar-month l-date))		42	(date (next d (1- tau)	
6	(leap-month (hindu-lunar-leap-month l-date))		43	(member (hindu-lunar-day-from-moment	
7	(day (hindu-lunar-day 1-date))		44	(hindu-sunrise d))	
8	(leap-day (hindu-lunar-leap-day l-date))		45	(list day (amod (1+ day) 30))))))	
9	(approx		46	(if leap-day (1+ date) date)))	
10	(+ hindu-epoch				
11	(* hindu-sidereal-year				
12	(+ year hindu-lunar-era		1	(defconstant ujjain	(20.25)
13	(/ (1- month) 12)))))		2	;; TYPE location	
14	(s (floor		3	;; Location of Ujjain.	
15	(- approx		4	(location (angle 23 9 0) (angle 75 46 6)	
16	(* hindu-sidereal-year		5	(mt 0) (hr (+ 5 461/9000))))	
17	(mod3 (- (/ (hindu-solar-longitude approx)				
18	(deg 360))		1	(defconstant hindu-location	(20.26)
19	(/ (1- month) 12))		2	·· TYPE location	(20:20)
20	-1/2 1/2)))))		3	Location (Nijain) for determining Hindu calendar	
21	<pre>(k (hindu-lunar-day-from-moment (+ s (hr 6))))</pre>		4	uijain)	
22	(est		-		
23	(- s (- day)				
24	(cond		1	(defun hindu-ascensional-difference (date location)	(20.27)
25	((< 3 k 27) ; Not borderline case.		2	;; TYPE (fixed-date location) -> rational-angle	
26	k)		3	;; Difference between right and oblique ascension	
27	((let* ((mid ; Middle of preceding solar month.		4	;; of sun on date at location.	
28	(hindu-lunar-from-fixed		5	(let* ((sin_delta	
29	(- s 15))))		6	(* 1397/3438 ; Sine of inclination.	
30	(or ; In month starting near s.		7	(hindu-sine (hindu-tropical-longitude date))))	

8	(phi (latitude location))		3	;; Sidereal daily motion of sun on date.	
9	(diurnal-radius		4	(let* ((mean-motion ; Mean daily motion in degrees.	
10	(hindu-sine (+ (deg 90) (hindu-arcsin sin_delta))))		5	(/ (deg 360) hindu-sidereal-year))	
11	(tan_phi ; Tangent of latitude as rational number.		6	(anomaly	
12	(/ (hindu-sine phi)		7	(hindu-mean-position date hindu-anomalistic-year))	
13	(hindu-sine (+ (deg 90) phi))))		8	(epicycle ; Current size of epicycle.	
14	(earth-sine (* sin_delta tan_phi)))		9	(- 14/360 (/ (abs (hindu-sine anomaly)) 1080)))	
15	(hindu-arcsin (- (/ earth-sine diurnal-radius)))))		10	(entry (quotient anomaly (angle 0 225 0)))	
			11	(sine-table-step ; Marginal change in anomaly	
			12	(- (hindu-sine-table (1+ entry))	
1	(defun hindu-tropical-longitude (date)	(20.28)	13	(hindu-sine-table entry)))	
2	;; TYPE fixed-date -> rational-angle		14	(factor	
3	;; Hindu tropical longitude on fixed date.		15	(* -3438/225 sine-table-step epicycle)))	
4	;; Assumes precession with maximum of 27 degrees		16	(* mean-motion (1+ factor))))	
5	;; and period of 7200 sidereal years				
6	;; (= 1577917828/600 days).		1	(defun hindu-rising-sign (date)	(20.31)
7	(let* ((days (- date hindu-epoch)) ; Whole days.		2	;; TYPE fixed-date -> rational-amplitude	()
8	(precession		3	;; Tabulated speed of rising of current zodiacal sign on	
9	(- (deg 27)		4	;; date.	
10	(abs		5	(let* ((i ; Index.	
11	(* (deg 108)		6	(quotient (hindu-tropical-longitude date)	
12	(mod3 (- (* 600/1577917828 days)		7	(deg 30))))	
13	1/4)		8	(nth (mod i 6)	
14	-1/2 1/2))))))		9	(list 1670/1800 1795/1800 1935/1800 1935/1800	
15	(mod (- (hindu-solar-longitude date) precession)		10	1795/1800 1670/1800))))	
16	360)))				
			1	(defun hindu-equation-of-time (date)	(20.32)
			2	:: TYPE fixed-date -> rational-moment	()
1	(defun hindu-solar-sidereal-difference (date)	(20.29)	3	:: Time from true to mean midnight of <i>date</i> .	
2	;; TYPE fixed-date -> rational-angle		4	:: (This is a gross approximation to the correct value.)	
3	;; Difference between solar and sidereal day on date.		5	(let* ((offset (hindu-sine	
4	(* (hindu-daily-motion date) (hindu-rising-sign date)))		6	(hindu-mean-position	
			7	date	
			8	hindu-anomalistic-year)))	
1	(defun hindu-daily-motion (date)	(20.30)	9	(equation-sun ; Sun's equation of center	
2	;; TYPE fixed-date -> rational-angle		10	;; Arcsin is not needed since small	

11	(* offset (angle 57 18 0)		4	<pre>(let* ((date (fixed-from-moment tee))</pre>	
12	(- 14/360 (/ (abs offset) 1080)))))		5	(time (time-from-moment tee))	
13	(* (/ (hindu-daily-motion date) (deg 360))		6	(q (floor (* 4 time))) ; quarter of day	
14	(/ equation-sun (deg 360))		7	(a (cond ((= q 0) ; early this morning	
15	hindu-sidereal-year)))		8	(hindu-sunset (1- date)))	
			9	((= q 3) ; this evening	
			10	(hindu-sunset date))	
1	(defun hindu-sunrise (date)	(20.33)	11	(t ; daytime today	
2	;; TYPE fixed-date -> rational-moment	. ,	12	(hindu-sunrise date))))	
3	;; Sunrise at hindu-location on <i>date</i> .		13	<pre>(b (cond ((= q 0) (hindu-sunrise date))</pre>	
4	(+ date (hr 6) ; Mean sunrise.		14	((= q 3) (hindu-sunrise (1+ date)))	
5	(/ (- (longitude ujjain) (longitude hindu-location))		15	(t (hindu-sunset date)))))	
6	(deg 360)) ; Difference from longitude.		16	(+ a (* 2 (- b a) (- time	
7	(- (hindu-equation-of-time date)) ; Apparent midnight.		17	(cond ((= q 3) (hr 18))	
8	(* ; Convert sidereal angle to fraction of civil day.		18	$((= q \ 0) \ (hr \ -6))$	
9	(/ 1577917828/1582237828 (deg 360))		19	(t (hr 6))))))))	
10	(+ (hindu-ascensional-difference date hindu-location)				
11	(* 1/4 (hindu-solar-sidereal-difference date))))))		1	(defun hindu-fullmoon-from-fixed (date)	(20.36)
			2	;; TYPE fixed-date -> hindu-lunar-date	
			3	;; Hindu lunar date, full-moon scheme,	
1	(defun hindu-sunset (date)	(20.34)	4	;; equivalent to fixed date.	
2	;; TYPE fixed-date -> rational-moment	. ,	5	(let* ((l-date (hindu-lunar-from-fixed date))	
3	;; Sunset at hindu-location on date.		6	(year (hindu-lunar-year l-date))	
4	(+ date (hr 18) ; Mean sunset.		7	(month (hindu-lunar-month l-date))	
5	(/ (- (longitude ujjain) (longitude hindu-location))		8	(leap-month (hindu-lunar-leap-month l-date))	
6	(deg 360)) ; Difference from longitude.		9	(day (hindu-lunar-day l-date))	
7	(- (hindu-equation-of-time date)) ; Apparent midnight.		10	(leap-day (hindu-lunar-leap-day l-date))	
8	(* ; Convert sidereal angle to fraction of civil day.		11	(m (if (>= day 16)	
9	(/ 1577917828/1582237828 (deg 360))		12	(hindu-lunar-month	
10	(+ (- (hindu-ascensional-difference date hindu-location))		13	(hindu-lunar-from-fixed (+ date 20)))	
11	(* 3/4 (hindu-solar-sidereal-difference date))))))		14	month)))	
			15	(hindu-lunar-date year m leap-month day leap-day)))	
1	(defun hindu-standard-from-sundial (tee)	(20.35)	1	(defun fixed-from-hindu-fullmoon (l-date)	(20.37)
2	;; TYPE rational-moment -> rational-moment		2	;; TYPE hindu-lunar-date -> fixed-date	
3	;; Hindu local time of temporal moment tee.		3	;; Fixed date equivalent to Hindu lunar <i>l-date</i>	

4 5 7 8 9	<pre>;; in full-moon scheme. (let* ((year (hindu-lunar-year l-date)) (month (hindu-lunar-month l-date)) (leap-month (hindu-lunar-leap-month l-date)) (day (hindu-lunar-day l-date)) (leap-day (hindu-lunar-leap-day l-date)) (m (cond ((or leap-month (<= day 15))</pre>		1 2 3 4 5	<pre>(defun ayanamsha (tee) ;; TYPE moment -> angle ;; Difference between tropical and sidereal solar longitude. (- (solar-longitude tee) (sidereal-solar-longitude tee)))</pre>	(20.40)
11 12 13 14 15 16	<pre>month) ((hindu-expunged? year (amod (1- month) 12)) (amod (- month 2) 12)) (t (amod (1- month) 12))))) (fixed-from-hindu-lunar (hindu-lunar-date year m leap-month day leap-day))))</pre>		1 2 3 4 5	<pre>(defconstant sidereal-start ;; TYPE angle (precession (universal-from-local (mesha-samkranti (ce 285)) hindu-location)))</pre>	(20.41)
1 2 3 4 5	<pre>(defun hindu-expunged? (1-year 1-month) ;; TYPE (hindu-lunar-year hindu-lunar-month) -> ;; TYPE boolean ;; True of Hindu lunar month 1-month in 1-year :: is expunded.</pre>	(20.38)	1 2 3 4	<pre>(defun astro-hindu-sunset (date) ;; TYPE fixed-date -> moment ;; Geometrical sunset at Hindu location on date. (dusk date hindu-location (deg 0)))</pre>	(20.42)
6 7 8 9 10	<pre>(/= 1-month (hindu-lunar-month (hindu-lunar-from-fixed (fixed-from-hindu-lunar (list 1-year 1-month false 15 false))))))</pre>		1 2 3 4 5	<pre>(defun sidereal-zodiac (tee) ;; TYPE moment -> hindu-solar-month ;; Sidereal zodiacal sign of the sun, as integer in range ;; 112, at moment <i>tee</i>. (1+ (quotient (sidereal-solar-longitude tee) (deg 30))))</pre>	(20.43)
1 2 3 4 5 6 7	<pre>(defun alt-hindu-sunrise (date) ;; TYPE fixed-date -> rational-moment ;; Astronomical sunrise at Hindu location on date, ;; per Lahiri, ;; rounded to nearest minute, as a rational number. (let* ((rise (dawn date hindu-location (angle 0 47 0)))) (* 1/24 1/60 (round (* rise 24 60)))))</pre>	(20.39)	1 2 3 4 5 6 7	<pre>(defun astro-hindu-calendar-year (tee) ;; TYPE moment -> hindu-solar-year ;; Astronomical Hindu solar year KY at given moment tee. (round (- (/ (- tee hindu-epoch)</pre>	(20.44)
1	(defun astro-hindu-solar-from-fixed (date)	(20.45)	16	month))))	
----	---	---------	----	--	---------
2	;; TYPE fixed-date -> hindu-solar-date		17	(+ start day -1)))	
3	;; Astronomical Hindu (Tamil) solar date equivalent to				
4	;; fixed date.				
5	(let* ((critical ; Sunrise on Hindu date.		1	(defun astro-lunar-day-from-moment (tee)	(20.47)
6	(astro-hindu-sunset date))		2	:: TYPE moment -> hindu-lunar-day	()
7	(month (sidereal-zodiac critical))		3	;; Phase of moon (tithi) at moment <i>tee</i> , as an integer in	
8	(year (- (astro-hindu-calendar-year critical)		4	:: the range 130.	
9	hindu-solar-era))		5	(1+ (quotient (lunar-phase tee) (deg 12))))	
10	(approx ; 3 days before start of mean month.				
11	(- date 3				
12	<pre>(mod (floor (sidereal-solar-longitude critical))</pre>		1	(define action binds, linear from fined (deta)	(20.49)
13	(deg 30))))		1	(defun astro-nindu-iunar-irom-iixed (date)	(20.48)
14	(start ; Search forward for beginning		2	;; IIPE IIXed-date -> Hindd-Iunar-date	
15	(next i approx ; of month.		3	;; Astronomical Allad funal date equivalent to fixed date.	
16	(= (sidereal-zodiac (astro-hindu-sunset i))		4	(let* ((critical	
17	month)))		5	(art-mindu-sumise date)) ; sumise that day.	
18	(day (- date start -1)))		7	(uay	
19	(hindu-solar-date year month day)))		0	(astro-runar-day-rrom-moment critical); bay or month	
			0	(leap-day ; ii previous day the same.	
			10	(= day (astro-runar-day-rrom-moment	
1	(dofum fixed-from-astro-hindu-solar (s-dato)	(20.46)	10	(last-new-moon	
2	(deful fixed-fiom-astro-findu-solar (s-date)	(20.40)	12	(now-moon-before critical))	
2	, Fixed data corresponding to Astronomical		12	(new-moon	
4	Hindu solar date (Tamil rule: Saka era)		14	(new-moon-at-or-after critical))	
5	(let+ ((month (standard-month s-date))		15	(solar-month · Solar month name	
6	(day (standard-day s-date))		16	(sidereal-zodiac last-new-moon))	
7	(vear (standard-vear s-date))		17	(leap-month · If begins and ends in same sign	
8	(approx · 3 days before start of mean month		18	(reap month (sidereal-zodiac pext-new-moon)))	
9	(+ hindu-enoch -3		19	(month · Month of lunar year	
10	(floor (* (+ (+ vear hindu-solar-era)		20	(amod (1+ solar-month) 12))	
11	(/ (1 - month) 12))		21	(vear : Solar year at end of month.	
12	mean-sidereal-year))))		22	(- (astro-hindu-calendar-year	
13	(start : Search forward for beginning		23	(if (<= month 2) : date might precede solar	
14	(next i approx ; of month.		24	; new year.	
				(-] - 100)	

26	date))		32	(and (hindu-lunar-leap-month mid)	
27	hindu-lunar-era)))		33	<pre>(not leap-month))))</pre>	
28	(hindu-lunar-date year month leap-month day leap-day)))		34	(mod3 k -15 15))	
			35	(t ; In preceding month.	
			36	(mod3 k 15 45)))))	
			37	(tau ; Refined estimate.	
1	(defun fixed-from-astro-hindu-lunar (l-date)	(20.49)	38	(- est (mod3 (- (astro-lunar-day-from-moment	
2	;; TYPE hindu-lunar-date -> fixed-date		39	(+ est (hr 6)))	
3	;; Fixed date corresponding to Hindu lunar date <i>l-date</i> .		40	day)	
4	(let* ((year (hindu-lunar-year l-date))		41	-15 15)))	
5	(month (hindu-lunar-month l-date))		42	(date (next d (1- tau)	
6	(leap-month (hindu-lunar-leap-month l-date))		43	(member (astro-lunar-day-from-moment	
7	(day (hindu-lunar-day l-date))		44	(alt-hindu-sunrise d))	
8	(leap-day (hindu-lunar-leap-day l-date))		45	(list day (amod (1+ day) 30))))))	
9	(approx		46	(if leap-day (1+ date) date)))	
10	(+ hindu-epoch				
11	(* mean-sidereal-year				
12	(+ year hindu-lunar-era			(defun hindu geler lengitude et er efter (lembde tee)	(20.50)
13	(/ (1- month) 12)))))		2	(defuil initial-solar-iongitude-at-of-atter (lambda tee)	(20.50)
14	(s (floor		2	;; IIPE (season moment) -> moment	
15	(- approx		3	;; when Windy solar longitude will be lambda degrees	
16	(* hindu-sidereal-year		5	(lot+ ((tau + Estimato (within 5 days)	
17	(mod3 (- (/ (sidereal-solar-longitude approx)		6	(ieta ((tau , Estimate (within) days).	
18	(deg 360))		7	(+ Lee)	
19	(/ (1- month) 12))		,	(* minut-sidereal-year (/ i (deg 500))	
20	-1/2 1/2)))))		0		
21	<pre>(k (astro-lunar-day-from-moment (+ s (hr 6))))</pre>		10	(2 (max + 20) (- + 20) E)), $A = 20 a fter + 20$	
22	(est		10	(a (max tee (= tau 5))); At of after tee.	
23	(- s (- day)		12	(b (+ tau 5)))	
24	(cond		12	(interval closed a b))))	
25	((< 3 k 27) ; Not borderline case.		15	(incerval-closed a b))))	
26	k)				
27	((let* ((mid ; Middle of preceding solar month.				
28	(astro-hindu-lunar-from-fixed		1	(defun mesha-samkranti (g-year)	(20.51)
29	(- s 15))))		2	;; TYPE gregorian-year -> rational-moment	
30	(or ; In month starting near s.		3	;; Fixed moment of Mesha samkranti (Vernal equinox)	
31	(/= (hindu-lunar-month mid) month)		4	;; in Gregorian g-year.	

5	(let* ((jan1 (gregorian-new-year g-year)))		16	(if (or (< new-moon critical)	
6	(hindu-solar-longitude-at-or-after (deg 0) jan1)))		17	(= (hindu-lunar-day-from-moment	
			18	(hindu-sunrise (1+ h-day))) 2))	
			19	0 1))))	
1	(defun hindu-lunar-day-at-or-after (k tee)	(20.52)			
2	;; TYPE (rational rational-moment) -> rational-moment		1	(defun hindu-lunar-on-or-before? (l-date1 l-date2)	(20.54)
3	;; Time lunar-day (tithi) number k begins at or after		2	;; TYPE (hindu-lunar-date hindu-lunar-date) -> boolean	
4	;; moment tee. k can be fractional (for karanas).		3	;; True if Hindu lunar date <i>l-date1</i> is on or before	
5	(let* ((phase ; Degrees corresponding to k.		4	;; Hindu lunar date 1-date2.	
6	(* (1- k) (deg 12)))		5	<pre>(let* ((month1 (hindu-lunar-month l-date1))</pre>	
7	(tau ; Mean occurrence of lunar-day.		6	(month2 (hindu-lunar-month l-date2))	
8	(+ tee (* (/ 1 (deg 360))		7	(leap1 (hindu-lunar-leap-month l-date1))	
9	(mod (- phase (hindu-lunar-phase tee))		8	(leap2 (hindu-lunar-leap-month l-date2))	
10	360)		9	(day1 (hindu-lunar-day l-date1))	
11	hindu-synodic-month)))		10	(day2 (hindu-lunar-day l-date2))	
12	(a (max tee (- tau 2)))		11	(leap-day1 (hindu-lunar-leap-day l-date1))	
13	(b (+ tau 2)))		12	(leap-day2 (hindu-lunar-leap-day l-date2))	
14	(invert-angular hindu-lunar-phase phase		13	(year1 (hindu-lunar-year l-date1))	
15	(interval-closed a b))))		14	(year2 (hindu-lunar-year l-date2)))	
			15	(or (< year1 year2)	
			16	(and (= year1 year2)	
1	(defun hindu-lunar-new-year (g-year)	(20.53)	17	(or (< month1 month2)	
2	:: TYPE gregorian-year -> fixed-date	(18	(and (= month1 month2)	
3	:: Fixed date of Hindu lunisolar new year in Gregorian		19	(or (and leap1 (not leap2))	
4	;; g-year.		20	(and (equal leap1 leap2)	
5	(let* ((jan1 (gregorian-new-year g-year))		21	(or (< day1 day2)	
6	(mina ; Fixed moment of solar longitude 330.		22	(and (= day1 day2)	
7	(hindu-solar-longitude-at-or-after (deg 330) jan1))		23	(or (not leap-day1)	
8	(new-moon ; Next new moon.		24	<pre>leap-day2)))))</pre>	
9	(hindu-lunar-day-at-or-after 1 mina))		25))))))	
10	(h-day (floor new-moon))				
11	(critical ; Sunrise that day.		1	(defun hindu-date-occur (l-year l-month l-day)	(20.55)
12	(hindu-sunrise h-day)))		2	;; TYPE (hindu-lunar-year hindu-lunar-month	
13	(+ h-day		3	;; TYPE hindu-lunar-day) -> fixed-date	
14	;; Next day if new moon after sunrise,		4	;; Fixed date of occurrence of Hindu lunar l-month,	
15	;; unless lunar day ends before next sunrise.		5	;; 1-day in Hindu lunar year 1-year, taking leap and	

568 Lisp Implementation

6	;; expunged days into account. When the month is	1	(defun diwali (g-year)	(20.57)
7	;; expunged, then the following month is used.	2	;; TYPE gregorian-year -> list-of-fixed-dates	
8	(let* ((lunar (hindu-lunar-date l-year l-month false	3	;; List of fixed date(s) of Diwali in Gregorian year	
9	l-day false))	4	;; g-year.	
10	(try (fixed-from-hindu-lunar lunar))	5	(hindu-lunar-holiday 8 1 g-year))	
11	(mid (hindu-lunar-from-fixed			
12	(if (> 1-day 15) (- try 5) try)))			
13	(expunged? (/= l-month (hindu-lunar-month mid)))	1	(defun hindu-tithi-occur (l-month tithi tee l-vear)	(20.58)
14	(l-date ; day in next month	2	:: TYPE (hindu-lunar-month rational rational	(
15	(hindu-lunar-date (hindu-lunar-year mid)	3	:: TYPE hindu-lunar-vear) -> fixed-date	
16	(hindu-lunar-month mid)	4	:: Fixed date of occurrence of Hindu lunar tithi prior	
17	(hindu-lunar-leap-month mid)	5	:: to sundial time tee, in Hindu lunar 1-month, 1-year.	
18	l-day false)))	6	(let* ((approx	
19	(cond (expunged?	7	(hindu-date-occur l-year l-month (floor tithi)))	
20	(1- (next d try	8	(lunar	
21	(not	9	(hindu-lunar-day-at-or-after tithi (- approx 2)))	
22	(hindu-lunar-on-or-before?	10	(try (fixed-from-moment lunar))	
23	<pre>(hindu-lunar-from-fixed d) l-date)))))</pre>	11	(tee h (standard-from-sundial (+ try tee) ujjain)))	
24	((/= l-day (hindu-lunar-day	12	(if (or (<= lunar tee h)	
25	(hindu-lunar-from-fixed try)))	13	(> (hindu-lunar-phase	
26	(1- try))	14	(standard-from-sundial (+ try 1 tee) ujjain))	
27	(t try))))	15	(* 12 tithi)))	
		16	try	
		17	(1+ try))))	
1	(defun hindu-lunar-holiday (l-month l-day g-year)	(20.56)		
2	;; TYPE (hindu-lunar-month hindu-lunar-day			
3	;; TYPE gregorian-year) -> list-of-fixed-dates	1	(defun hindu-lunar-event (l-month tithi tee g-year)	(20.59)
4	;; List of fixed dates of occurrences of Hindu lunar	2	;; TYPE (hindu-lunar-month rational rational	
5	;; month, day in Gregorian year g-year.	3	;; TYPE gregorian-year) -> list-of-fixed-dates	
6	(let* ((l-year (hindu-lunar-year	4	;; List of fixed dates of occurrences of Hindu lunar tithi	
7	(hindu-lunar-from-fixed	5	;; prior to sundial time tee, in Hindu lunar 1-month,	
8	(gregorian-new-year g-year))))	6	;; in Gregorian year g-year.	
9	(date0 (hindu-date-occur l-year l-month l-day))	7	(let* ((l-year (hindu-lunar-year	
10	(date1 (hindu-date-occur (1+ l-year) l-month l-day)))	8	(hindu-lunar-from-fixed	
11	(list-range (list date0 date1)	9	(gregorian-new-year g-year))))	
12	(gregorian-year-range g-year))))	10	(date0 (hindu-tithi-occur l-month tithi tee l-year))	

11	(date1 (hindu-tithi-occur		1	(defun yoga (date)	(20.64)
12	<pre>l-month tithi tee (1+ l-year))))</pre>		2	;; TYPE fixed-date -> 1-27	
13	(list-range (list date0 date1)		3	;; Hindu yoga on <i>date</i> .	
14	(gregorian-year-range g-year))))		4	(1+ (floor (mod (/ (+ (hindu-solar-longitude date)	
			5	(hindu-lunar-longitude date))	
			6	(angle 0 800 0))	
1	(defun shiva (g-year)	(20.60)	7	27))))	
2	;; TYPE gregorian-year -> list-of-fixed-dates				
3	;; List of fixed date(s) of Night of Shiva in Gregorian				
4	;; year g-year.				
5	(hindu-lunar-event 11 29 (hr 24) g-year))		1	(defun sacred-wednesdays (g-year)	(20.65)
			2	;; TYPE gregorian-year -> list-of-fixed-dates	
			3	;; List of Wednesdays in Gregorian year g-year	
1	(defun rama (g-year)	(20.61)	4	;; that are day 8 of Hindu lunar months.	
2	;; TYPE gregorian-year -> list-of-fixed-dates		5	(sacred-wednesdays-in-range	
3	;; List of fixed date(s) of Rama's Birthday in Gregorian		6	(gregorian-year-range g-year)))	
4	;; year g-year.				
5	(hindu-lunar-event 1 9 (hr 12) g-year))				
			1	(defun sacred-wednesdays-in-range (range)	(20.66)
1	(defun hindu-lunar-station (date)	(20.62)	2	;; TYPE range -> list-of-fixed-dates	
2	;; TYPE fixed-date -> nakshatra		3	;; List of Wednesdays within <i>range</i> of dates	
3	;; Hindu lunar station (nakshatra) at sunrise on date.		4	;; that are day 8 of Hindu lunar months.	
4	(let* ((critical (hindu-sunrise date)))		5	(let* ((a (begin range))	
5	(1+ (quotient (hindu-lunar-longitude critical)		6	(b (end range))	
6	(angle 0 800 0)))))		7	(wed (kday-on-or-after wednesday a))	
			8	<pre>(h-date (hindu-lunar-from-fixed wed)))</pre>	
			9	(if (in-range? wed range)	
1	(defun karana (n)	(20.63)	10	(append	
2	;; TYPE 1-60 -> 0-10		11	(if (= (hindu-lunar-day h-date) 8)	
3	:: Number (0-10) of the name of the <i>n</i> -th (1-60) Hindu		12	(list wed)	
4	;; karana.		13	nil)	
5	(cond ((= n 1) 0))		14	(sacred-wednesdays-in-range	
6	((> n 57) (- n 50))		15	(interval (1+ wed) b)))	
7	(t (amod (1- n) 7))))		16	nil)))	

570 Lisp Implementation

D.21 The Tibetan Calendar

	D.21 The Tibetan Calendar		1	(defun tibetan-sun-equation (alpha)	(21.2)
1	(defun tibetan-date (year month leap-month day leap-day)		2	;; TYPE rational-angle -> rational	
2	:: TYPE (tibetan-year tibetan-month		3	;; Interpolated tabular sine of solar anomaly alpha.	
3	;; TYPE tibetan-leap-month tibetan-day		4	(cond ((> alpha 6) (- (tibetan-sun-equation (- alpha 6))))	
4	:: TYPE tibetan-leap-day) -> tibetan-date		5	((> alpha 3) (tibetan-sun-equation (- 6 alpha)))	
5	(list year month leap-month day leap-day))		6	((integerp alpha)	
	·		7	(nth alpha (list (mins 0) (mins 6) (mins 10) (mins 11))))
			8	(t (+ (* (mod alpha 1)	
1	(defun tibetan-year (date)		9	(tibetan-sun-equation (ceiling alpha)))	
2	;; TYPE tibetan-date -> tibetan-year		10	(* (mod (- alpha) 1)	
3	(first date))		11	<pre>(tibetan-sun-equation (floor alpha))))))</pre>	
1	(defun tibetan-month (date)		1	(defun tibetan-moon-equation (alpha)	(21.3)
2	;; TYPE tibetan-date -> tibetan-month		2	;; TYPE rational-angle -> rational	
3	(second date))		3	;; Interpolated tabular sine of lunar anomaly alpha.	
			4	(cond ((> alpha 14) (- (tibetan-moon-equation (- alpha 14))))	
			5	((> alpha 7) (tibetan-moon-equation (- 14 alpha)))	
1	(defun tibetan-leap-month (date)		6	((integerp alpha)	
2	;; TYPE tibetan-date -> tibetan-leap-month		7	(nth alpha	
3	(third date))		8	(list (mins 0) (mins 5) (mins 10) (mins 15)	
			9	(mins 19) (mins 22) (mins 24) (mins 25))))	
			10	(t (+ (* (mod alpha 1)	
1	(defun tibetan-day (date)		11	(tibetan-moon-equation (ceiling alpha)))	
2	;; TYPE tibetan-date -> tibetan-day		12	(* (mod (- alpha) 1)	
3	(fourth date))		13	<pre>(tibetan-moon-equation (floor alpha))))))</pre>	
1	(defun tibetan-leap-day (date)		1	(defun fixed-from-tibetan (t-date)	(21.4)
2	;; TYPE tibetan-date -> tibetan-leap-day		2	;; TYPE tibetan-date -> fixed-date	
3	(fifth date))		3	;; Fixed date corresponding to Tibetan lunar date t-date.	
			4	(let* ((year (tibetan-year t-date))	
			5	(month (tibetan-month t-date))	
1	(defconstant tibetan-epoch	(21.1)	6	(leap-month (tibetan-leap-month t-date))	
2	;; TYPE fixed-date		7	(day (tibetan-day t-date))	
3	(fixed-from-gregorian (gregorian-date -127 december 7)))		8	(leap-day (tibetan-leap-day t-date))	

9	(months ; Lunar month count.		20	(final	
10	(floor (+ (* 804/65 (1- year)) (* 67/65 month)		21	d (- est 2)	
11	(if leap-month -1 0) 64/65)))		22	(>= date	
12	(days ; Lunar day count.		23	(fixed-from-tibetan	
13	(+ (* 30 months) day))		24	<pre>(tibetan-date year0 month0 false d false)))))</pre>	
14	(mean ; Mean civil days since epoch.		25	(leap-month (> day0 30))	
15	(+ (* days 11135/11312) -30		26	(day (amod day0 30))	
16	(if leap-day 0 -1) 1071/1616))		27	(month (amod (cond ((> day day0) (1- month0))	
17	(solar-anomaly		28	(leap-month (1+ month0))	
18	(mod (+ (* days 13/4824) 2117/4824) 1))		29	(t month0))	
19	(lunar-anomaly		30	12))	
20	(mod (+ (* days 3781/105840) 2837/15120) 1))		31	(year (cond ((and (> day day0) (= month0 1))	
21	(sun (- (tibetan-sun-equation (* 12 solar-anomaly))))		32	(1- year0))	
22	<pre>(moon (tibetan-moon-equation (* 28 lunar-anomaly))))</pre>		33	((and leap-month (= month0 12))	
23	(floor (+ tibetan-epoch mean sun moon))))		34	(1+ year0))	
			35	(t year0)))	
			36	(leap-day	
		(21.5)	37	(= date	
1	(derun tibetan-irom-iixed (date)	(21.5)	38	(fixed-from-tibetan	
2	;; TYPE fixed-date -> tibetan-date		39	(tibetan-date year month leap-month day true)))))	
3	;; Tibetan lunar date corresponding to fixed date.		40	(tibetan-date year month leap-month day leap-day)))	
4	(let* ((cap-Y (+ 365 49/5/18382)) ; Average Tibetan year.				
5	(years (ceiling (/ (- date tibetan-epoch) cap-Y)))				
6	(year); Search for year.		1	(defun tibetan-leap-month? (t-year t-month)	(21.6)
,	(linal y years		2	;; TYPE (tibetan-year tibetan-month) -> boolean	. ,
0	(>- uale		3	;; True if t -month is leap in Tibetan year t -year.	
9	(fixed=ffom=tibetan		4	(= t-month	
10	(month) . Course for month		5	(tibetan-month	
12	(final m 1		6	(tibetan-from-fixed	
12	()dato		7	(fixed-from-tibetan	
13	(>= uale (fixed_from_tibetan		8	(tibetan-date t-year t-month true 2 false))))))	
14	(fixed=ffom=tibetan				
15	(cost , Retimated day				
17	(= data (fived-from-tibetan		1	(defun tibetan-lean-day2 (t-year t-menth t-day)	(21.7)
18	(tibetan-date year) month(falso 1 falso))))		2	TYDE (tibetan-year tibetan-month tibetan-day) -> booloan	(21.7)
10	(daw) . Soarch for day		2	, True if t-day is leap in Tibetan	
19	(day); search for day.		3	;; ifue if c-day is reap in ilbecan	

572 Lisp Implementation

4	;; month <i>t-month</i> and year <i>t-year</i> .	6	(fixed-from-tibetan	
5	(or	7	(tibetan-date t-year 1 t-leap 1 false))))	
6	(= t-day			
7	(tibetan-day			
8	(tibetan-from-fixed	1	(defun tibetan-new-vear (g-vear)	(21.9)
9	(fixed-from-tibetan	2	:: TYPE gregorian-year -> list-of-fixed-dates	(,
10	(tibetan-date t-year t-month false t-day true)))))	3	;; List of fixed dates of Tibetan New Year in	
11	;; Check also in leap month if there is one.	4	;; Gregorian year g-year.	
12	(= t-day	5	(let* ((dec31 (gregorian-year-end g-year))	
13	(tibetan-day	6	(t-year (tibetan-year (tibetan-from-fixed dec31))))	
14	(tibetan-from-fixed	7	(list-range	
15	(fixed-from-tibetan	8	(list (losar (1- t-year))	
16	(tibetan-date t-year t-month	9	(losar t-year))	
17	(tibetan-leap-month? t-year t-month)	10	(gregorian-year-range g-year))))	
18	t-day true)))))))			

1 (defun losar (t-year)

2

3

4

5

;; TYPE tibetan-year -> fixed-date

;; in Tibetan year t-year.

;; Fixed date of Tibetan New Year (Losar)

(let* ((t-leap (tibetan-leap-month? t-year 1)))

(21.8)

References

- N. Dershowitz and E. M. Reingold, "Modulo Intervals: A Proposed Notation," ACM SIGACT News, vol. 43, no. 3, pp. 60–64, 2012.
- [2] G. L. Steele, Jr., Common LISP: The Language, 2nd edn., Digital Press, Bedford, MA, 1990.

Downloaded from https://www.cambridge.org/core. Access paid by the UCSB Libraries, on 26 Mar 2018 at 07:44:25, subject to the Cambridge Core terms of use, available at https://www.cambridge.org/core/terms. https://doi.org/10.1017/9781107415058.029



In octo libros Deemendatione temporum Index.

A

A 70	- concertant land
A^{B}	76.c.105.a.179.0.378.0
1 1 Abulani	14) 4.3/0.4.3/9.0
Achos Gun Ochos	3300 0
Actions pute Octoos	224.0
Alliaca vicioria	237.4
A. D. IIII. EID.	-6446 1-0 1
Adar	70.0.165.0.378.0.379.0
Adarpahajent	143.0.378.0.379.0 1
Adejer	230.0
Adorare de geniculu	340.0
Adrianus Imper.	244.4
Adu	310.4
Aegon	171.4 379.0
Aequatio anni Gracoru	m 23.0
Aeraquid	235.4.236.4.313.0
Aera Hupanica	234.4
Aera tres technica	367.0
Acra Coptica velmarty	rum 245.6 c
Acquinoctium	190.0
Acquinoctialia punctar	non ii dem stellus semper
affixa	181.0
Acquinoctiorum epochei	mmobilis 181.4
Acquinoctium autumna	leobseruatum a Muha-
mede Albateni	192.4
ab Hipparcho	193.0
Acquinoctium tempore 1	Viceni consessus 21. Mar-
tij	193.0
Acthiopum lingua proxi	me abest a Chaldra &
Allyria	338 d
Aethiopes quando ineunt	iciunia 342.d
Agon Capitelinus	243.0
Abalibaigueras	285.4
Abali	378.d
Allelenmening	ibid.
Atalcomentos	303.0
A LEXT PS (Will a	Gelalutatus 226.4
Alexander, Imperator I	m 41.4.42.4
eins juoi ne commenta	it 209.6
quo anno Darium on	12.6.209.0
eiusobitus	225.6
Autenjispugna	204.4.382.d
Amalekicopiaacieta	284 d
Amistris & Ester, idem	164 \$ 370.6
Amfchir	104.0.3/ 9.0
and once needed	40.0

Annas pontifex	253.6
Annius Viterbiensis notatur	215 d
Annus	7.d.294.c
Annus Saturni	7.d
Annus Luna	ibid.
Anni duo pracipua genera apud vete	res 8.6
Anni principium naturale & popular	e 25.4.6
Annus non ab ea die institutus qua m	undus con-
ditus est. led ab ea qua Sol & Luna	109.4
Annus defectiuus, abundans, comm	nunis er a-
auabilis 10.4.11.4.86.c.	127.4.315.6
Annus cauus er plenus	9.d.315.C
Annus embolimaus	298.0
Annus Colaris	7.4
Anni Colaris genera	II.d
Annus Calaris ante Calarem coonitus	50.4
Annus folaris Indaicus	167.d
. Annus Colaris Hipparchi	11.6
Annus Lunaris	0.d.70.b
periodicus en limplex	o.d
Annus I unaris equabilis in duas bas	tes aquales
diuiditur	317.d
Anno lunari quo tempore Iudei vi	i cæperunt
70 4	1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1
Annus colectis	11.d. 180.c
eiusexaminatio	190.6
. Anni coleft hemerologium	382
Annus caleftis Dianyfianus	170.0
Orientalium	172.4
Annus colectis messurilionas ionue	owns 180.4
duogenera	ibid.c
Annus equabily minor Grecorum	15.4
Annus Atticus 362 dierum	27.0
eius initium incurrit in tempus br	Hma 25.4
trincipium populare ab Hecatomi	aone.natu-
rale à Gameliane	ibid.
Annie tatius Grecie modus	17.6
Anno Greco equabili due caltigatio	mes adhibi-
Anno Graco agricora una orgo Brit	8.d
Ammus Bom morum Tulianus I	ss.a. & inde
Annus Iulianus est fundamentum	inftituti au-
Stanie	14.6
Annue Romanorum Tiltus	116.c.117.d
anins Aunanderim menfium fuit	ibid.
Annue Tilianus	429.6
21/1/H3 L104/1/H3	

First page of the index to Joseph Scaliger's *De Emendatione Temporum* (Frankfort edition, 1593). (Courtesy of the University of Illinois, Urbana, IL.)