

PYCALCAL – Literate Calendars in PYTHON

Enrico Spinielli¹

December 30, 2009

¹I want to thank my wife, Gilda, for the patience of having looked after all kids and things while I was *playing* with this project. I also want to thank Prof. Reingold for his his prompt reply to all my questions. Finally my gratitude goes to my parents, Alida and Mario, for the sacrifice they endeavoured in order to allow me get inspired by Science.

Contents

1	Literate Calendrical Calculations	1
2	The Calendars	6
3	Future evolutions	164
4	Technicalities	170
	Bibliography	188
A	Chunks	190

Chapter 1

Literate Calendrical Calculations

THIS document describes a PYTHON [6] implementation of the calendrical algorithms described in the book *Calendrical Calculations* [1]. According to the authors, the book is the companion text of the algorithms implemented in the COMMON LISP package CC3. The source code for CC3, `calendrica-3.0.c1`, is made available electronically by the publisher, see [1].

I provide full reference to the original COMMON LISP and credit (and deep respect) to its authors, Prof.s Dershowitz and Reingold.

On the (long) way to complete this project, I experimented with Scons [10], Test-Driven Development [11] and web applications [2].

I tackled this task the Literate Programming way [3] using the NOWEB tool. [8, 7] As such all code and documentation is generated from a single source file and prefixed with the following warning:

```
1 <generated code warning 1>≡
  AUTOMATICALLY GENERATED FROM pycalcal.nw: ANY CHANGES WILL BE OVERWRITTEN.
```

This code is used in chunks 4, 9, 49, 58, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 111, 114, 117, 122, 125, 128, 131, 134, 137, 188, 191, 193, 194, 197, 203, and 234.

1.1 Structure

The software is for now a single piece of text (a.k.a. a PYTHON file):

```
2 <* 2>≡
  <pycalcal.py 3>
```

Root chunk (not used in this document).

It is organised as follows:

3 *<pysical.py 3>*≡
 <testa 9>
 <global import statements 11>
 <basic code 13>
 <egyptian and armenian calendars 65>
 <gregorian calendar 51>
 <julian calendar 59>
 <iso calendar 68>
 <coptic and ethiopic calendars 71>
 <ecclesiastical calendars 74>
 <islamic calendar 77>
 <hebrew calendar 80>
 <mayan calendars 83>
 <old hindu calendars 86>
 <balinese calendar 89>
 <time and astronomy 92>
 <astronomical lunar calendars 95>
 <persian calendar 112>
 <bahai calendar 115>
 <french revolutionary calendar 118>
 <chinese calendar 123>
 <modern hindu calendars 126>
 <tibetan calendar 129>
 <coda 133>

This code is used in chunk 2.

There is as well a companion file with unit tests inspired by the examples spread in the book [1], its Appendix C or devised by myself.

```

4  <pycalcaltests.py 4>≡
    # <generated code warning 1>
    <LICENSE 7>
    from pycalcal import *
    from appendixCUnitTest import AppendixCTable1TestCaseBase
    from appendixCUnitTest import AppendixCTable2TestCaseBase
    from appendixCUnitTest import AppendixCTable3TestCaseBase
    from appendixCUnitTest import AppendixCTable4TestCaseBase
    from appendixCUnitTest import AppendixCTable5TestCaseBase
    import unittest

    <basic code unit test 50>
    <egyptian and armenian calendars unit test 66>
    <gregorian calendar unit test 57>
    <iso calendar unit test 69>
    <julian calendar unit test 60>
    <coptic and ethiopic calendars unit test 72>
    <ecclesiastical calendars unit test 75>
    <islamic calendar unit test 78>
    <hebrew calendar unit test 81>
    <mayan calendars unit test 84>
    <old hindu calendars unit test 87>
    <balinese calendar unit test 90>
    <time and astronomy unit test 94>
    <persian calendar unit test 113>
    <bahai calendar unit test 116>
    <french revolutionary calendar unit test 119>
    <chinese calendar unit test 124>
    <modern hindu calendars unit test 127>
    <tibetan calendar unit test 130>
    <astronomical lunar calendars unit test 135>

    <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139, AppendixCTable2TestCaseBase 155, AppendixCTable3TestCaseBase 164, AppendixCTable4TestCaseBase 175, and AppendixCTable5TestCaseBase 184.

The code for tests execution is:

```

5  <execute tests 5>≡
    if __name__ == "__main__":
        unittest.main()

```

This code is used in chunks 4, 49, 58, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 111, 114, 117, 122, 125, 128, 131, 134, and 137.

Given I also want to be able to run a single unit tests part in isolation, i.e. when working on Persian calendar I just want to run the Persian calendar's tests, I generate a unit tests file per each part. Here is an example of how the unit tests are templated for an hypothetical xyzy calendar/section:

```
<<xyzyUnitTest.py>>=
# <<generated code warning>>
<<import for testing>>
from appendixCUnitTest import AppendixCTable1TestCaseBase
<<xyzy unit test>>
<<execute tests>>
```

The following imports are to be used:

```
6 <import for testing 6>≡
    from pycalcal import *
    import unittest
```

This code is used in chunks 49, 58, 64, 67, 70, 73, 76, 79, 82, 85, 88, 91, 111, 114, 117, 122, 125, 128, 131, 134, and 137.

1.2 Copyright and Legalise

My copyright notice is intended to make this work accessible and free to everybody for any type of use (i.e. commercial, educational ...). I took an MIT License.

```
7 <LICENSE 7>≡
    # Copyright (c) 2009 Enrico Spinielli
    #
    # Permission is hereby granted, free of charge, to any person obtaining a copy
    # of this software and associated documentation files (the "Software"), to deal
    # in the Software without restriction, including without limitation the rights
    # to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
    # copies of the Software, and to permit persons to whom the Software is
    # furnished to do so, subject to the following conditions:
    #
    # The above copyright notice and this permission notice shall be included in
    # all copies or substantial portions of the Software.
    #
    # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    # IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
    # FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
    # AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
    # LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
    # OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
    # THE SOFTWARE.
```

This code is used in chunks 4 and 9.

But remember that Dershowitz and Reingold have Copyrighted their algorithms!

```
8 <copyright Dershowitz and Reingold 8>≡
    # PLEASE READ THE COPYRIGHT IN THE FILE THAT INSPIRED THIS WORK
    # see lines 6-80 in calendrica-3.0.c1
```

Root chunk (not used in this document).

9 $\langle \text{testa } 9 \rangle \equiv$
 `"""Python implementation of Dershowitz and Reingold's 'Calendrica Calculations'.`
 `Python implementation of calendrical algorithms as described in Common`
 `Lisp in calendrical-3.0.cl (and errata as made available by the authors.)`
 `The companion book is Dershowitz and Reingold 'Calendrica Calculations',`
 `3rd Ed., 2008, Cambridge University Press.`
 `License: MIT License for my work, but read the one`
 `for calendrica-3.0.cl which inspired this work.`
 `Author: Enrico Spinielli`
 `"""`
 $\langle \text{LICENSE } 7 \rangle$
 `# $\langle \text{generated code warning } 1 \rangle$`

This code is used in chunk 3.

The following is where I control the project version number in a centralised way. The version is number is composed of 3 integers separated by a dot, '' which stand for

`@<major version>.<minor version>.<increment>`

Then version number 0.9.2 means revision 2 of project version 0.9.

10 $\langle \text{project version } 10 \rangle \equiv$
 `1.0.0`

This code is used in chunk 203.

Chapter 2

The Calendars

2.1 Scaffolding

THE accuracy of the algorithms presented require definition of constants and calculations that span many bits. I enable true division feature as in PEP 238 [13] in order to smoothly express simple constants as defined in the COMMON LISP code, i.e. 1/360

```
11 <global import statements 11>≡  
    # use true division  
    from __future__ import division
```

This definition is continued in chunk 12.
This code is used in chunk 3.

In order to insure the same precision of computations as in COMMON LISP code where numbers are postfixed with L0, meaning 50-bit precision, I use the `mpmath` library [4] and set the precision accordingly floating-point arithmetic.

```
12 <global import statements 11>+≡  
    # Precision in bits, for places where CL postfixes numbers with L0, meaning  
    # at least 50 bits of precision  
    from mpmath import *  
    mp.prec = 50
```

This code is used in chunk 3.

2.2 Basics

The following are general definitions, algorithms and helper functions that will be used to perform a lot of the various calculations for the subsequent calendars.

2.2.1 Implementation

Some computations have meaningless results in certain circumstances, they will return a predefined value, `BOGUS`, to mark this case.

```
13 <basic code 13>≡
    #####
    # basic calendrical algorithms #
    #####
    # see lines 244-247 in calendrica-3.0.cl
    BOGUS = 'bogus'
```

This definition is continued in chunks 14-16, 19, 22, 25, 28, 31, 34, 37-40, 43, and 46-48.

This code is used in chunk 3.

Defines:

`BOGUS`, used in chunks 56, 80, 83, 92, 95, 123, and 186.

Some COMMON LISP functions are available under other names and/or in additional packages in PYTHON or with a different semantic, so they are aliased or loaded or (re)defined accordingly. This is the case for `quotient`, `floor` and `round`, the COMMON LISP versions return integers while in PYTHON they can return a float if (at least) one of the arguments is a float.

```
14 <basic code 13>+≡
    # see lines 249-252 in calendrica-3.0.cl
    # m // n
    # The following
    #     from operator import floordiv as quotient
    # is not ok, the corresponding CL code
    # uses CL 'floor' which always returns an integer
    # (the floating point equivalent is 'ffloor'), while
    # 'quotient' from operator module (or corresponding //)
    # can return a float if at least one of the operands
    # is a float...so I redefine it (and 'floor' and 'round' as well: in CL
    # they always return an integer.)
    def quotient(m, n):
        """Return the whole part of m/n."""
        from operator import floordiv
        return ifloor(m / n)
```

This code is used in chunk 3.

Defines:

`quotient`, used in chunks 55, 56, 59, 65, 68, 71, 74, 77, 80, 83, 86, 89, 112, 115, 120, 123, 126, and 132.

Uses `ifloor` 15.

For `floor` and `round` I decided to make it explicit the fact that they return an integer and named them with a prefixed *i* (for integer):

```
15 <basic code 13>+≡
    # I (re)define floor: in CL it always returns an integer.
    # I make it explicit the fact it returns an integer by
    # naming it ifloor
    def ifloor(n):
        """Return the whole part of m/n."""
        from math import floor
        return int(floor(n))

    # I (re)define round: in CL it always returns an integer.
    # I make it explicit the fact it returns an integer by
    # naming it iround
    def iround(n):
        """Return the whole part of m/n."""
        from __builtin__ import round
        return int(round(n))

    # m % n (this works as described in book for negative integres)
    # It is interesting to note that
    # mod(1.5, 1)
    # returns the decimal part of 1.5, so 0.5; given a moment 'm'
    # mod(m, 1)
    # returns the time of the day
    from operator import mod

    # see lines 254-257 in calendrica-3.0.cl
    def amod(x, y):
        """Return the same as a % b with b instead of 0."""
        return y + (mod(x, -y))
```

This code is used in chunk 3.

Defines:

`amod`, used in chunks 56, 61, 68, 83, 86, 89, 123, 126, and 129.
`ifloor`, used in chunks 14, 40, 46, 48, 86, 97, 112, 115, 120, 123, 126, 129, and 132.
`iround`, used in chunks 96, 109, 112, 115, 120, 123, 126, and 132.
`mod`, used in chunks 28, 38, 40, 46, 54, 55, 59, 65, 71, 74, 77, 80, 83, 86, 89, 92, 95, 97, 99, 109, 110, 112, 115, 120, 123, 126, 129, 132, 186, and 242.

The following definitions are a translation in PYTHON of COMMON LISP macros `Macro`. `Macro` definition is a very powerful tool but it does not exist in the PYTHON language.

The principles of the algorithms described in [1] is to quickly get an approximation and then refine it to get the correct result. `next` and `final` are used in the *refine* phase.

```
16 <basic code 13>+≡
    # see lines 259-264 in calendrica-3.0.cl
    def next(i, p):
        """Return first integer greater or equal to initial index, i,
        such that condition, p, holds."""
        return i if p(i) else next(i+1, p)
```

This code is used in chunk 3.

Defines:

`next`, used in chunks 18, 80, 109, 112, 115, 120, 123, 126, 132, 184, 193, and 200.

```
17 <basic code tests 17>≡
    <test next 18>
```

This definition is continued in chunks 20, 23, 26, 29, 32, 35, 41, and 44.

This code is used in chunk 50.

```

18  <test next 18>≡
    def testNext(self):
        self.assertEqual(next(0, lambda i: i == 3), 3)
        self.assertEqual(next(0, lambda i: i == 0), 0)

```

This code is used in chunk 17.
 Uses next 16.

```

19  <basic code 13>+≡
    # see lines 266-271 in calendrica-3.0.c1
    def final(i, p):
        """Return last integer greater or equal to initial index, i,
        such that condition, p, holds."""
        return i - 1 if not p(i) else final(i+1, p)

```

This code is used in chunk 3.
 Defines:
 final, used in chunks 21, 80, 109, and 129.

```

20  <basic code tests 17>+≡
    <test final 21>

```

This code is used in chunk 50.

```

21  <test final 21>≡
    def testFinal(self):
        self.assertEqual(final(0, lambda i: i == 3), -1)
        self.assertEqual(final(0, lambda i: i < 3), 2)
        self.assertEqual(final(0, lambda i: i < 0), -1)

```

This code is used in chunk 20.
 Defines:
 testFinal, never used.
 Uses final 19.

The following functions are used mainly in the astronomical algorithms in order to evaluate polynomial approximations of equations of motion.

```

22  <basic code 13>+≡
    # see lines 273-281 in calendrica-3.0.c1
    def summa(f, k, p):
        """Return the sum of f(i) from i=k, k+1, ... till p(i) holds true or 0.
        This is a tail recursive implementation."""
        return 0 if not p(k) else f(k) + summa(f, k+1, p)

    def altsumma(f, k, p):
        """Return the sum of f(i) from i=k, k+1, ... till p(i) holds true or 0.
        This is an implementation of the Summation formula from Kahan,
        see Theorem 8 in Goldberg, David 'What Every Computer Scientist
        Should Know About Floating-Point Arithmetic', ACM Computer Survey,
        Vol. 23, No. 1, March 1991."""
        if not p(k):
            return 0
        else:
            S = f(k)
            C = 0
            j = k + 1
            while p(j):
                Y = f(j) - C
                T = S + Y
                C = (T - S) - Y
                S = T
                j += 1

            return S

```

This code is used in chunk 3.

Defines:

altsumma, used in chunk 24.
summa, used in chunks 24 and 80.

```

23  <basic code tests 17>+≡
    <test summa 24>

```

This code is used in chunk 50.

```

24  <test summa 24>≡
    def testSumma(self):
        self.assertEqual(summa(lambda x: 1, 1, lambda i: i<=4), 4)
        self.assertEqual(summa(lambda x: 1, 0, lambda i: i>=4), 0)
        self.assertEqual(summa(lambda x: x**2, 1, lambda i: i<=4), 30)

    def testAltSumma(self):
        # I should add more tests with floating point arithmetic...
        self.assertEqual(altsumma(lambda x: 1.0, 1, lambda i: i<=4), 4)
        self.assertEqual(altsumma(lambda x: 1.0, 0, lambda i: i>=4), 0)
        self.assertEqual(altsumma(lambda x: x**2, 1, lambda i: i<=4), 30)

```

This code is used in chunk 23.

Defines:

testAltSumma, never used.
testSumma, never used.

Uses **altsumma** 22 and **summa** 22.

`binary_search` is looking for a value of a function within an interval given a precision criteria to satisfy.

```
25 <basic code 13>+≡
    # see lines 283-293 in calendrica-3.0.c1
    def binary_search(lo, hi, p, e):
        """Bisection search for x in [lo, hi] such that condition 'e' holds.
        p determines when to go left."""
        x = (lo + hi) / 2
        if p(lo, hi):
            return x
        elif e(x):
            return binary_search(lo, x, p, e)
        else:
            return binary_search(x, hi, p, e)
```

This code is used in chunk 3.

Defines:

`binary_search`, used in chunks 27, 28, 92, and 126.

```
26 <basic code tests 17>+≡
    <test binary search 27>
```

This code is used in chunk 50.

```
27 <test binary search 27>≡
    def testBinarySearch(self):
        fminusy = lambda x, y: fx(x) - y
        p = lambda a, b: abs(fminusy(0.5 * (a+b), y)) <= 10**-5
        e = lambda x: fminusy(x, y) >= 0
        # function y = f(x), f(x) = x, y0 = 1.0; solution is x0 = 1.0
        fx = lambda x: x
        y = 1.0
        x0 = 1.0
        self.assertTrue(binary_search(0.0, 3.1, p, e) - x0 <= 10 ** -5)
        # new function y = f(x), f(x) = x**2 - 4*x + 4, y0 = 0.0; solution x0=2.0
        y = 0.0
        x0 = 2.0
        fx = lambda x: x**2 - 4 * x + 4.0
        self.assertTrue(binary_search(1.5, 2.5, p, e) - x0 <= 10 ** -5)
```

This code is used in chunk 26.

Defines:

`testBinarySearch`, never used.

Uses `binary_search` 25.

```
28 <basic code 13>+≡
    # see lines 295-302 in calendrica-3.0.c1
    def invert_angular(f, y, a, b, prec=10**-5):
        """Find inverse of angular function 'f' at 'y' within interval [a,b].
        Default precision is 0.00001"""
        return binary_search(a, b,
                               (lambda l, h: ((h - l) <= prec)),
                               (lambda x: mod((f(x) - y), 360) < 180))
    #def invert_angular(f, y, a, b):
    #    from scipy.optimize import brentq
    #    return(brentq((lambda x: mod(f(x) - y), 360)), a, b, xtol=error)
```

This code is used in chunk 3.

Defines:

`invert_angular`, used in chunks 30, 99, 109, and 126.

Uses `binary_search` 25, `interval` 47, and `mod` 15.

29 \langle basic code tests 17 $\rangle + \equiv$
 \langle test invert angular 30 \rangle

This code is used in chunk 50.

30 \langle test invert angular 30 $\rangle \equiv$

```
def testInvertAngular(self):
    from math import tan, radians
    # find angle theta such that tan(theta) = 1
    # assert that theta - pi/4 <= 10**-5
    self.assertTrue(invert_angular(tan,
                                   1.0,
                                   0,
                                   radians(60.0)) - radians(45.0) <= 10**-5)
```

This code is used in chunk 29.

Defines:

testInvertAngular, never used.

Uses angle 92 and invert_angular 28 28.

sigma is used expecially for astronomical formulas in order to go thru tables of value, see 2.14.3. This implementation is really horrible...but I will not touch it untill I will have time and courage to go thru understanding it again!

31 \langle basic code 13 $\rangle + \equiv$

```
# see lines 304-313 in calendrica-3.0.c1
def sigma(l, b):
    """Return the sum of body 'b' for indices i1..in
    running simultaneously thru lists l1..ln.
    List 'l' is of the form [[i1 l1]..[in ln]]"""
    # 'l' is a list of 'n' lists of the same lenght 'L' [l1, l2, l3, ...]
    # 'b' is a lambda with 'n' args
    # 'sigma' sums all 'L' applications of 'b' to the relevant tuple of args
    # >> a = [ 1, 2, 3, 4]
    # >> b = [ 5, 6, 7, 8]
    # >> c = [ 9,10,11,12]
    # >> l = [a,b,c]
    # >> z = zip(*l)
    # >> z
    # [(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
    # >> b = lambda x, y, z: x * y * z
    # >> b(*z[0]) # apply b to first elem of i
    # 45
    # >> temp = []
    # >> z = zip(*l)
    # >> for e in z: temp.append(b(*e))
    # >> temp
    # [45, 120, 231, 384]
    # >> from operator import add
    # >> reduce(add, temp)
    # 780
    return sum(b(*e) for e in zip(*l))
```

This code is used in chunk 3.

Defines:

sigma, used in chunks 33, 97, 109, and 110.

32 \langle basic code tests 17 $\rangle + \equiv$
 \langle test sigma 33 \rangle

This code is used in chunk 50.

```

33  <test sigma 33>≡
    def testSigma(self):
        a = [ 1, 2, 3, 4]
        b = [ 5, 6, 7, 8]
        c = [ 9,10,11,12]
        ell = [a,b,c]
        bi = lambda x, y, z: x * y * z
        self.assertEqual(sigma(ell, bi), 780)

```

This code is used in chunk 32.
 Defines:
 testSigma, never used.
 Uses sigma 31.

```

34  <basic code 13>+≡
    # see lines 315-321 in calendrica-3.0.cl
    from copy import copy
    def poly(x, a):
        """Calculate polynomial with coefficients 'a' at point x.
        The polynomial is a[0] + a[1] * x + a[2] * x^2 + ...a[n-1]x^(n-1)
        the result is
        a[0] + x(a[1] + x(a[2] + ...+ x(a[n-1]))...)"""
        # This implementation is also known as Horner's Rule.
        n = len(a) - 1
        p = a[n]
        for i in range(1, n+1):
            p = p * x + a[n-i]
        return p

```

This code is used in chunk 3.
 Defines:
 poly, used in chunks 36, 92, 97, 99, 101, 103, 105, 107, 109, and 110.

```

35  <basic code tests 17>+≡
    <test poly 36>

```

This code is used in chunk 50.

```

36  <test poly 36>≡
    def testPoly(self):
        self.assertEqual(poly(0, [2, 2, 1]), 2)
        self.assertEqual(poly(1, [2, 2, 1]), 5)

```

This code is used in chunk 35.
 Defines:
 testPoly, never used.
 Uses poly 34.

Now it is time to begin with calendars: let's define the instant in time when time is counted from:

```
37 <basic code 13>+≡
    # see lines 323-329 in calendrica-3.0.c1
    # Epoch definition. I took it out explicitly from rd().
    def epoch():
        """Epoch definition. For Rata Diem, R.D., it is 0 (but any other reference
        would do.)"""
        return 0

    def rd(tee):
        """Return rata diem (number of days since epoch) of moment in time, tee."""
        return tee - epoch()
```

This code is used in chunk 3.

Defines:

epoch, used in chunks 80 and 86.

rd, used in chunks 38, 48, 52, 65, 68, 84, 94, 123, 139, 141, 143, 145, 147, 148, 150, 151, 154, 155, 157, 159, 161, 163, 164, 166, 168, 170, 172, 174, 175, 177, 179, 181, 183, 184, 186, and 243.

And here are some other basilar (and arbitrary) definitions: days of the week, date and time data structures.

The days of the week constants are defined as constants from 1 to 7, where Sunday is (arbitrarily) assigned 0.

```
38 <basic code 13>+≡
    # $
    # see lines 331-334 in calendrica-3.0.cl
    SUNDAY = 0

    # see lines 10-15 in calendrica-3.0.errata.cl
    MONDAY = 1

    # see lines 17-20 in calendrica-3.0.errata.cl
    TUESDAY = 2

    # see lines 22-25 in calendrica-3.0.errata.cl
    WEDNESDAY = 3

    # see lines 27-30 in calendrica-3.0.errata.cl
    THURSDAY = 4

    # see lines 32-35 in calendrica-3.0.errata.cl
    FRIDAY = 5

    # see lines 37-40 in calendrica-3.0.errata.cl
    SATURDAY = SUNDAY + 6

    DAYS_OF_WEEK_NAMES = {
        SUNDAY : "Sunday",
        MONDAY : "Monday",
        TUESDAY : "Tuesday",
        WEDNESDAY : "Wednesday",
        THURSDAY : "Thursday",
        FRIDAY : "Friday",
        SATURDAY : "Saturday"}

    # see lines 366-369 in calendrica-3.0.cl
    def day_of_week_from_fixed(date):
        """Return day of the week from a fixed date 'date'."""
        return mod(date - rd(0) - SUNDAY, 7)
```

This code is used in chunk 3.

Defines:

day_of_week_from_fixed, used in chunks 56, 68, 80, and 141.

FRIDAY, used in chunks 56, 80, and 139.

MONDAY, used in chunks 56, 80, 81, and 139.

SATURDAY, used in chunks 80, 81, and 139.

SUNDAY, used in chunks 56, 68, 74, 80, 132, and 139.

THURSDAY, used in chunks 68, 80, 81, and 139.

TUESDAY, used in chunks 56, 80, 81, and 139.

WEDNESDAY, used in chunks 80, 81, 126, and 139.

Uses mod 15 and rd 37.

```

39  <basic code 13>+≡
    # see lines 371-374 in calendrica-3.0.cl
    def standard_month(date):
        """Return the month of date 'date'."""
        return date[1]

    # see lines 376-379 in calendrica-3.0.cl
    def standard_day(date):
        """Return the day of date 'date'."""
        return date[2]

    # see lines 381-384 in calendrica-3.0.cl
    def standard_year(date):
        """Return the year of date 'date'."""
        return date[0]

    # see lines 386-388 in calendrica-3.0.cl
    def time_of_day(hour, minute, second):
        """Return the time of day data structure."""
        return [hour, minute, second]

    # see lines 390-392 in calendrica-3.0.cl
    def hour(clock):
        """Return the hour of clock time 'clock'."""
        return clock[0]

    # see lines 394-396 in calendrica-3.0.cl
    def minute(clock):
        """Return the minutes of clock time 'clock'."""
        return clock[1]

    # see lines 398-400 in calendrica-3.0.cl
    def seconds(clock):
        """Return the seconds of clock time 'clock'."""
        return clock[2]

```

This code is used in chunk 3.

Defines:

- hour, used in chunks 40, 42, 43, and 95-97.
- minute, used in chunks 40, 42, 43, 96, and 126.
- seconds, used in chunks 42, 43, 46, 92, and 96.
- standard_day, used in chunks 55, 56, 59, 61, 65, 71, 77, 80, 86, 112, 120, 126, and 132.
- standard_month, used in chunks 55, 56, 59, 61, 65, 71, 77, 80, 86, 112, 120, 126, and 132.
- standard_year, used in chunks 55, 56, 59, 61, 63, 65, 71, 77, 80, 86, 112, 120, 126, 132, and 145.
- time_of_day, used in chunk 40.

The following functions convert from moment to fixed date, extract the time of the day from a moment.

```
40 <basic code 13>+≡
    # see lines 402-405 in calendrica-3.0.cl
    def fixed_from_moment(tee):
        """Return fixed date from moment 'tee'."""
        return ifloor(tee)

    # see lines 407-410 in calendrica-3.0.cl
    def time_from_moment(tee):
        """Return time from moment 'tee'."""
        return mod(tee, 1)

    # see lines 412-419 in calendrica-3.0.cl
    def clock_from_moment(tee):
        """Return clock time hour:minute:second from moment 'tee'."""
        time = time_from_moment(tee)
        hour = ifloor(time * 24)
        minute = ifloor(mod(time * 24 * 60, 60))
        second = mod(time * 24 * 60 * 60, 60)
        return time_of_day(hour, minute, second)
```

This code is used in chunk 3.

Defines:

- clock_from_moment, used in chunks 42 and 96.
- fixed_from_moment, used in chunks 92, 95, and 126.
- time_from_moment, used in chunk 92.

Uses hour 39, ifloor 15, minute 39, mod 15, and time_of_day 39.

```
41 <basic code tests 17>+≡
    <test clock from moment 42>
```

This code is used in chunk 50.

```
42 <test clock from moment 42>≡
    def testClockFromMoment(self):
        c = clock_from_moment(3.5)
        self.assertEqual(hour(c), 12)
        self.assertEqual(minute(c), 0)
        self.assertAlmostEqual(seconds(c), 0, 2)

        c = clock_from_moment(3.75)
        self.assertEqual(hour(c), 18)
        self.assertEqual(minute(c), 0)
        self.assertAlmostEqual(seconds(c), 0, 2)

        c = clock_from_moment(3.8)
        self.assertEqual(hour(c), 19)
        self.assertEqual(minute(c), 11)
        self.assertAlmostEqual(seconds(c), 59.9999, 2)
```

This code is used in chunk 41.

Defines:

- testClockFromMoment, never used.

Uses clock_from_moment 40, hour 39, minute 39, and seconds 39.

```

43  <basic code 13>+≡
    # see lines 421-427 in calendrica-3.0.cl
    def time_from_clock(hms):
        """Return time of day from clock time 'hms'."""
        h = hour(hms)
        m = minute(hms)
        s = seconds(hms)
        return(1/24 * (h + ((m + (s / 60)) / 60)))

```

This code is used in chunk 3.

Defines:

time_from_clock, used in chunks 45 and 96.

Uses hour 39, minute 39, and seconds 39.

```

44  <basic code tests 17>+≡
    <test time from clock 45>

```

This code is used in chunk 50.

```

45  <test time from clock 45>≡
    def testTimeFromClock(self):
        self.assertAlmostEqual(time_from_clock([12, 0, 0]), 0.5, 2)
        self.assertAlmostEqual(time_from_clock([18, 0, 0]), 0.75, 2)
        self.assertAlmostEqual(time_from_clock([19, 12, 0]), 0.8, 2)

```

This code is used in chunk 44.

Defines:

testTimeFromClock, never used.

Uses time_from_clock 43.

Here we define the angular data structure and relative helper functions.

```

46  <basic code 13>+≡
    # see lines 429-431 in calendrica-3.0.cl
    def degrees_minutes_seconds(d, m, s):
        """Return the angular data structure."""
        return [d, m, s]

    # see lines 433-440 in calendrica-3.0.cl
    def angle_from_degrees(alpha):
        """Return an angle in degrees:minutes:seconds from angle,
        'alpha' in degrees."""
        d = ifloor(alpha)
        m = ifloor(60 * mod(alpha, 1))
        s = mod(alpha * 60 * 60, 60)
        return degrees_minutes_seconds(d, m, s)

```

This code is used in chunk 3.

Defines:

angle_from_degrees, never used.

degrees_minutes_seconds, never used.

Uses angle 92, degrees 92, ifloor 15, mod 15, and seconds 39.

These helper functions are used to deal with intervals and ranges of events:

```
47 <basic code 13>+≡
# see lines 502-510 in calendrica-3.0.cl
def list_range(ell, range):
    """Return those moments in list ell that occur in range 'range'."""
    return filter(lambda x: is_in_range(x, range), ell)

# see lines 482-485 in calendrica-3.0.cl
def interval(t0, t1):
    """Return the range data structure."""
    return [t0, t1]

# see lines 487-490 in calendrica-3.0.cl
def start(range):
    """Return the start of range 'range'."""
    return range[0]

# see lines 492-495 in calendrica-3.0.cl
def end(range):
    """Return the end of range 'range'."""
    return range[1]

# see lines 497-500 in calendrica-3.0.cl
def is_in_range(tee, range):
    """Return True if moment 'tee' falls within range 'range',
    False otherwise."""
    return start(range) <= tee <= end(range)
```

This code is used in chunk 3.

Defines:

- end, used in chunks 56, 89, 95, 118, 123, and 126.
- interval, used in chunks 28, 56, 89, and 126.
- is_in_range, used in chunks 56 and 126.
- list_range, used in chunks 63, 71, 77, 80, 126, and 129.
- start, used in chunks 56, 80, 89, 95, 115, 118, 123, 126, 132, and 186.

Julian days are of basic importance especially in Astronomy.

```
48 <basic code 13>+≡
# see lines 442-445 in calendrica-3.0.cl
JD_EPOCH = rd(mpf(-1721424.5))

# see lines 447-450 in calendrica-3.0.cl
def moment_from_jd(jd):
    """Return the moment corresponding to the Julian day number 'jd'."""
    return jd + JD_EPOCH

# see lines 452-455 in calendrica-3.0.cl
def jd_from_moment(tee):
    """Return the Julian day number corresponding to moment 'tee'."""
    return tee - JD_EPOCH

# see lines 457-460 in calendrica-3.0.cl
def fixed_from_jd(jd):
    """Return the fixed date corresponding to Julian day number 'jd'."""
    return ifloor(moment_from_jd(jd))

# see lines 462-465 in calendrica-3.0.cl
def jd_from_fixed(date):
    """Return the Julian day number corresponding to fixed date 'rd'."""
    return jd_from_moment(date)

# see lines 467-470 in calendrica-3.0.cl
MJD_EPOCH = rd(678576)

# see lines 472-475 in calendrica-3.0.cl
def fixed_from_mjd(mjd):
    """Return the fixed date corresponding to modified Julian day 'mjd'."""
    return mjd + MJD_EPOCH

# see lines 477-480 in calendrica-3.0.cl
def mjd_from_fixed(date):
    """Return the modified Julian day corresponding to fixed date 'rd'."""
    return date - MJD_EPOCH
```

This code is used in chunk 3.

Defines:

- fixed_from_jd, used in chunks 65, 83, 89, and 143.
- fixed_from_mjd, used in chunk 143.
- JD_EPOCH, never used.
- jd_from_fixed, used in chunk 143.
- jd_from_moment, never used.
- MJD_EPOCH, never used.
- mjd_from_fixed, used in chunk 143.
- moment_from_jd, never used.

Uses ifloor 15 and rd 37.

2.2.2 Unit tests

The relevant tests are available to be run in isolation in basicCodeUnitTest.py

```
49 <basicCodeUnitTest.py 49>≡
# <generated code warning 1>
<import for testing 6>
from appendixCUnitTest import AppendixCTable1TestCaseBase
<basic code unit test 50>
<execute tests 5>
```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139.

and grouped together so that they can be included in `pycalcaltests.py`

```
50 <basic code unit test 50>≡  
    class BasicCodeTestCase(unittest.TestCase):  
        <basic code tests 17>
```

This definition is continued in chunk 140.

This code is used in chunks 4 and 49.

2.3 Gregorian Calendar

```
51 <gregorian calendar 51>≡  
    #####  
    # gregorial calendar algorithms #  
    #####  
    <gregorian date and epoch 52>  
    <gregorian months 53>  
    <gregorian leap year function 54>  
    <gregorian conversion functions 55>  
    <gregorian year start and end 56>
```

This code is used in chunk 3.

```
52 <gregorian date and epoch 52>≡  
    # see lines 586-589 in calendrica-3.0.cl  
    def gregorian_date(year, month, day):  
        """Return a Gregorian date data structure."""  
        return [year, month, day]  
  
    # see lines 591-595 in calendrica-3.0.cl  
    GREGORIAN_EPOCH = rd(1)
```

This code is used in chunk 51.

Defines:

`gregorian_date`, used in chunks 56, 57, 59, 68, 74, 92, 96-98, 115, 118, 119, 123, 129, 139, 164, and 191.

`GREGORIAN_EPOCH`, used in chunks 55, 56, and 74.

Uses `rd` 37.

```

53  <gregorian months 53>≡
    # see lines 597-600 in calendrica-3.0.cl
    JANUARY = 1

    # see lines 602-605 in calendrica-3.0.cl
    FEBRUARY = 2

    # see lines 607-610 in calendrica-3.0.cl
    MARCH = 3

    # see lines 612-615 in calendrica-3.0.cl
    APRIL = 4

    # see lines 617-620 in calendrica-3.0.cl
    MAY = 5

    # see lines 622-625 in calendrica-3.0.cl
    JUNE = 6

    # see lines 627-630 in calendrica-3.0.cl
    JULY = 7

    # see lines 632-635 in calendrica-3.0.cl
    AUGUST = 8

    # see lines 637-640 in calendrica-3.0.cl
    SEPTEMBER = 9

    # see lines 642-645 in calendrica-3.0.cl
    OCTOBER = 10

    # see lines 647-650 in calendrica-3.0.cl
    NOVEMBER = 11

    # see lines 652-655 in calendrica-3.0.cl
    DECEMBER = 12

```

This code is used in chunk 51.

Defines:

APRIL, used in chunks 74, 98, and 123.
 AUGUST, used in chunks 71, 83, and 123.
 DECEMBER, used in chunks 56, 59, 63, 68, and 129.
 FEBRUARY, used in chunks 61, 86, 96, and 123.
 JANUARY, used in chunks 56, 57, 59, 92, 97, and 123.
 JULY, used in chunks 56, 61, 77, 97, and 123.
 JUNE, never used.
 MARCH, used in chunks 56, 57, 59, 61, 112, 115, and 123.
 MAY, used in chunks 56 and 61.
 NOVEMBER, used in chunks 56, 57, 60, and 62.
 OCTOBER, used in chunks 60, 61, 80, and 119.
 SEPTEMBER, used in chunks 56 and 118.

```

54  <gregorian leap year function 54>≡
    # see lines 657-663 in calendrica-3.0.cl
    def is_gregorian_leap_year(g_year):
        """Return True if Gregorian year 'g_year' is leap."""
        return (mod(g_year, 4) == 0) and (mod(g_year, 400) not in [100, 200, 300])

```

This code is used in chunk 51.

Defines:

is_gregorian_leap_year, used in chunks 55-57 and 115.
 Uses mod 15.


```

55  <gregorian conversion functions 55>≡
    # see lines 665-687 in calendrica-3.0.c1
    def fixed_from_gregorian(g_date):
        """Return the fixed date equivalent to the Gregorian date 'g_date'."""
        month = standard_month(g_date)
        day    = standard_day(g_date)
        year   = standard_year(g_date)
        return ((GREGORIAN_EPOCH - 1) +
                (365 * (year - 1)) +
                quotient(year - 1, 4) -
                quotient(year - 1, 100) +
                quotient(year - 1, 400) +
                quotient((367 * month) - 362, 12) +
                (0 if month <= 2
                 else (-1 if is_gregorian_leap_year(year) else -2)) +
                day)

    # see lines 689-715 in calendrica-3.0.c1
    def gregorian_year_from_fixed(date):
        """Return the Gregorian year corresponding to the fixed date 'date'."""
        d0 = date - GREGORIAN_EPOCH
        n400 = quotient(d0, 146097)
        d1 = mod(d0, 146097)
        n100 = quotient(d1, 36524)
        d2 = mod(d1, 36524)
        n4 = quotient(d2, 1461)
        d3 = mod(d2, 1461)
        n1 = quotient(d3, 365)
        year = (400 * n400) + (100 * n100) + (4 * n4) + n1
        return year if (n100 == 4) or (n1 == 4) else (year + 1)

```

This code is used in chunk 51.

Defines:

fixed_from_gregorian, used in chunks 56, 57, 59, 74, 92, 96, 98, 115, 118, 119, 123, 129, 145, and 191.

gregorian_year_from_fixed, used in chunks 56, 68, 80, 97, 112, 115, 123, 132, 145, and 168.

Uses GREGORIAN_EPOCH 52, is_gregorian_leap_year 54, mod 15, quotient 14 14, standard_day 39, standard_month 39, and standard_year 39.

```

56  (gregorian year start and end 56)≡
    # see lines 717-721 in calendrica-3.0.cl
    def gregorian_new_year(g_year):
        """Return the fixed date of January 1 in Gregorian year 'g_year'."""
        return fixed_from_gregorian(gregorian_date(g_year, JANUARY, 1))

    # see lines 723-727 in calendrica-3.0.cl
    def gregorian_year_end(g_year):
        """Return the fixed date of December 31 in Gregorian year 'g_year'."""
        return fixed_from_gregorian(gregorian_date(g_year, DECEMBER, 31))

    # see lines 729-733 in calendrica-3.0.cl
    def gregorian_year_range(g_year):
        """Return the range of fixed dates in Gregorian year 'g_year'."""
        return interval(gregorian_new_year(g_year), gregorian_year_end(g_year))

    # see lines 735-756 in calendrica-3.0.cl
    def gregorian_from_fixed(date):
        """Return the Gregorian date corresponding to fixed date 'date'."""
        year = gregorian_year_from_fixed(date)
        prior_days = date - gregorian_new_year(year)
        correction = (0
            if (date < fixed_from_gregorian(gregorian_date(year,
                                                                MARCH,
                                                                1)))
            else (1 if is_gregorian_leap_year(year) else 2))
        month = quotient((12 * (prior_days + correction)) + 373, 367)
        day = 1 + (date - fixed_from_gregorian(gregorian_date(year, month, 1)))
        return gregorian_date(year, month, day)

    # see lines 758-763 in calendrica-3.0.cl
    def gregorian_date_difference(g_date1, g_date2):
        """Return the number of days from Gregorian date 'g_date1'
        till Gregorian date 'g_date2'."""
        return fixed_from_gregorian(g_date2) - fixed_from_gregorian(g_date1)

    # see lines 42-49 in calendrica-3.0.errata.cl
    def day_number(g_date):
        """Return the day number in the year of Gregorian date 'g_date'."""
        return gregorian_date_difference(
            gregorian_date(standard_year(g_date) - 1, DECEMBER, 31),
            g_date)

    # see lines 53-58 in calendrica-3.0.cl
    def days_remaining(g_date):
        """Return the days remaining in the year after Gregorian date 'g_date'."""
        return gregorian_date_difference(
            g_date,
            gregorian_date(standard_year(g_date), DECEMBER, 31))

    # see lines 779-801 in calendrica-3.0.cl
    def alt_fixed_from_gregorian(g_date):
        """Return the fixed date equivalent to the Gregorian date 'g_date'.
        Alternative calculation."""
        month = standard_month(g_date)
        day = standard_day(g_date)
        year = standard_year(g_date)
        m = amod(month - 2, 12)
        y = year + quotient(month + 9, 12)
        return ((GREGORIAN_EPOCH - 1) +
            -306
            +

```

```

        365 * (y - 1)          +
        quotient(y - 1, 4)     +
        -quotient(y - 1, 100)  +
        quotient(y - 1, 400)   +
        quotient(3 * m - 1, 5) +
        30 * (m - 1)          +
        day)

# see lines 803-825 in calendrica-3.0.cl
def alt_gregorian_from_fixed(date):
    """Return the Gregorian date corresponding to fixed date 'date'.
    Alternative calculation."""
    y = gregorian_year_from_fixed(GREGORIAN_EPOCH - 1 + date + 306)
    prior_days = date - fixed_from_gregorian(gregorian_date(y - 1, MARCH, 1))
    month = amod(quotient(5 * prior_days + 2, 153) + 3, 12)
    year = y - quotient(month + 9, 12)
    day = date - fixed_from_gregorian(gregorian_date(year, month, 1)) + 1
    return gregorian_date(year, month, day)

# see lines 827-841 in calendrica-3.0.cl
def alt_gregorian_year_from_fixed(date):
    """Return the Gregorian year corresponding to the fixed date 'date'.
    Alternative calculation."""
    approx = quotient(date - GREGORIAN_EPOCH + 2, 146097/400)
    start = (GREGORIAN_EPOCH
             +
             (365 * approx)
             +
             quotient(approx, 4)
             +
             -quotient(approx, 100)
             +
             quotient(approx, 400))
    return approx if (date < start) else (approx + 1)

# see lines 843-847 in calendrica-3.0.cl
def independence_day(g_year):
    """Return the fixed date of United States Independence Day in
    Gregorian year 'g_year'."""
    return fixed_from_gregorian(gregorian_date(g_year, JULY, 4))

# see lines 849-853 in calendrica-3.0.cl
def kday_on_or_before(k, date):
    """Return the fixed date of the k-day on or before fixed date 'date'.
    k=0 means Sunday, k=1 means Monday, and so on."""
    return date - day_of_week_from_fixed(date - k)

# see lines 855-859 in calendrica-3.0.cl
def kday_on_or_after(k, date):
    """Return the fixed date of the k-day on or after fixed date 'date'.
    k=0 means Sunday, k=1 means Monday, and so on."""
    return kday_on_or_before(k, date + 6)

# see lines 861-865 in calendrica-3.0.cl
def kday_nearest(k, date):
    """Return the fixed date of the k-day nearest fixed date 'date'.
    k=0 means Sunday, k=1 means Monday, and so on."""
    return kday_on_or_before(k, date + 3)

# see lines 867-871 in calendrica-3.0.cl
def kday_after(k, date):
    """Return the fixed date of the k-day after fixed date 'date'.
```

```

    k=0 means Sunday, k=1 means Monday, and so on."""
    return kday_on_or_before(k, date + 7)

# see lines 873-877 in calendrica-3.0.cl
def kday_before(k, date):
    """Return the fixed date of the k-day before fixed date 'date'.
    k=0 means Sunday, k=1 means Monday, and so on."""
    return kday_on_or_before(k, date - 1)

# see lines 62-74 in calendrica-3.0.errata.cl
def nth_kday(n, k, g_date):
    """Return the fixed date of n-th k-day after Gregorian date 'g_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."""
    if n > 0:
        return 7*n + kday_before(k, fixed_from_gregorian(g_date))
    elif n < 0:
        return 7*n + kday_after(k, fixed_from_gregorian(g_date))
    else:
        return BOGUS

# see lines 892-897 in calendrica-3.0.cl
def first_kday(k, g_date):
    """Return the 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."""
    return nth_kday(1, k, g_date)

# see lines 899-904 in calendrica-3.0.cl
def last_kday(k, g_date):
    """Return the 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."""
    return nth_kday(k - 1, g_date)

# see lines 906-910 in calendrica-3.0.cl
def labor_day(g_year):
    """Return the fixed date of United States Labor Day in Gregorian
    year 'g_year' (the first Monday in September)."""
    return first_kday(MONDAY, gregorian_date(g_year, SEPTEMBER, 1))

# see lines 912-916 in calendrica-3.0.cl
def memorial_day(g_year):
    """Return the fixed date of United States' Memorial Day in Gregorian
    year 'g_year' (the last Monday in May)."""
    return last_kday(MONDAY, gregorian_date(g_year, MAY, 31))

# see lines 918-923 in calendrica-3.0.cl
def election_day(g_year):
    """Return the fixed date of United States' Election Day in Gregorian
    year 'g_year' (the Tuesday after the first Monday in November)."""
    return first_kday(TUESDAY, gregorian_date(g_year, NOVEMBER, 2))

# see lines 925-930 in calendrica-3.0.cl
def daylight_saving_start(g_year):
    """Return the fixed date of the start of United States daylight
    saving time in Gregorian year 'g_year' (the second Sunday in March)."""
    return nth_kday(2, SUNDAY, gregorian_date(g_year, MARCH, 1))

# see lines 932-937 in calendrica-3.0.cl
def daylight_saving_end(g_year):

```

```

        """Return the fixed date of the end of United States daylight saving
        time in Gregorian year 'g_year' (the first Sunday in November)."""
        return first_kday(SUNDAY, gregorian_date(g_year, NOVEMBER, 1))

# see lines 939-943 in calendrica-3.0.c1
def christmas(g_year):
    """Return the fixed date of Christmas in Gregorian year 'g_year'."""
    return fixed_from_gregorian(gregorian_date(g_year, DECEMBER, 25))

# see lines 945-951 in calendrica-3.0.c1
def advent(g_year):
    """Return the fixed date of Advent in Gregorian year 'g_year'
    (the Sunday closest to November 30)."""
    return kday_nearest(SUNDAY,
                        fixed_from_gregorian(gregorian_date(g_year,
                                                            NOVEMBER,
                                                            30)))

# see lines 953-957 in calendrica-3.0.c1
def epiphany(g_year):
    """Return the fixed date of Epiphany in U.S. in Gregorian year 'g_year'
    (the first Sunday after January 1)."""
    return first_kday(SUNDAY, gregorian_date(g_year, JANUARY, 2))

def epiphany_it(g_year):
    """Return fixed date of Epiphany in Italy in Gregorian year 'g_year'."""
    return gregorian_date(g_year, JANUARY, 6)

# see lines 959-974 in calendrica-3.0.c1
def unlucky_fridays_in_range(range):
    """Return the list of Fridays within range 'range' of fixed dates that
    are day 13 of the relevant Gregorian months."""
    a = start(range)
    b = end(range)
    fri = kday_on_or_after(FRIDAY, a)
    date = gregorian_from_fixed(fri)
    ell = [fri] if (standard_day(date) == 13) else []
    if is_in_range(fri, range):
        ell[:0] = unlucky_fridays_in_range(interval(fri + 1, b))
        return ell
    else:
        return []

```

This code is used in chunk 51.

Defines:

```

advent, never used.
alt_fixed_from_gregorian, used in chunk 145.
alt_gregorian_from_fixed, used in chunk 145.
alt_gregorian_year_from_fixed, used in chunk 145.
christmas, never used.
day_number, used in chunk 57.
daylight_saving_end, never used.
daylight_saving_start, never used.
days_remaining, never used.
election_day, never used.
epiphany, never used.
epiphany_it, never used.
first_kday, never used.
gregorian_date_difference, used in chunk 97.
gregorian_from_fixed, used in chunks 57, 145, and 168.
gregorian_new_year, used in chunks 63, 68, 71, 77, 80, 97, 123, 126, and 132.

```

gregorian_year_end, used in chunks 68 and 129.
 gregorian_year_range, used in chunks 63, 71, 77, 80, 89, 126, and 129.
 independence_day, never used.
 kday_after, used in chunks 74 and 132.
 kday_before, used in chunk 80.
 kday_nearest, never used.
 kday_on_or_after, used in chunk 126.
 kday_on_or_before, never used.
 labor_day, never used.
 last_kday, never used.
 memorial_day, never used.
 nth_kday, used in chunk 68.
 unlucky_fridays_in_range, never used.
 Uses amod 15, BOGUS 13, day_of_week_from_fixed 38, DECEMBER 53, end 47, fixed_from_gregorian 55, FRIDAY 38, gregorian_date 52, GREGORIAN_EPOCH 52, gregorian_year_from_fixed 55, interval 47, is_gregorian_leap_year 54, is_in_range 47, JANUARY 53, JULY 53, MARCH 53, MAY 53, MONDAY 38, NOVEMBER 53, quotient 14 14, SEPTEMBER 53, standard_day 39, standard_month 39, standard_year 39, start 47, SUNDAY 38, and TUESDAY 38.

```

57  <gregorian calendar unit test 57>≡
    class GregorianCalendarSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347
            self.aDate = gregorian_date(1945, NOVEMBER, 12)
            self.myDate = gregorian_date(1967, JANUARY, 30)
            self.aLeapDate = gregorian_date(1900, MARCH, 1)

        def testConversionFromFixed(self):
            self.assertEqual(
                gregorian_from_fixed(self.testvalue), self.aDate)

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_gregorian(self.aDate))

        def testLeapYear(self):
            self.assertTrue(is_gregorian_leap_year(2000))
            self.assertTrue(not is_gregorian_leap_year(1900))

        def testDayNumber(self):
            self.assertEqual(day_number(self.myDate), 30)
            self.assertEqual(day_number(self.aLeapDate), 60)

```

This definition is continued in chunk 144.

This code is used in chunks 4 and 58.

Defines:

GregorianCalendarSmokeTestCase, never used.

Uses day_number 56, fixed_from_gregorian 55, gregorian_date 52, gregorian_from_fixed 56, is_gregorian_leap_year 54, JANUARY 53, MARCH 53, and NOVEMBER 53.

2.3.1 Unit tests

As usual the unit tests are as follows

```

58  <gregorianCalendarUnitTest.py 58>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable1TestCaseBase
    <gregorian calendar unit test 57>
    <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139.

2.4 Julian Calendar

```
59 (julian calendar 59)≡
#####
# julian calendar algorithms #
#####
# see lines 1037-1040 in calendrica-3.0.cl
def julian_date(year, month, day):
    """Return the Julian date data structure."""
    return [year, month, day]

# see lines 1042-1045 in calendrica-3.0.cl
JULIAN_EPOCH = fixed_from_gregorian(gregorian_date(0, DECEMBER, 30))

# see lines 1047-1050 in calendrica-3.0.cl
def bce(n):
    """Return a negative value to indicate a BCE Julian year."""
    return -n

# see lines 1052-1055 in calendrica-3.0.cl
def ce(n):
    """Return a positive value to indicate a CE Julian year."""
    return n

# see lines 1057-1060 in calendrica-3.0.cl
def is_julian_leap_year(j_year):
    """Return True if Julian year 'j_year' is a leap year in
    the Julian calendar."""
    return mod(j_year, 4) == (0 if j_year > 0 else 3)

# see lines 1062-1082 in calendrica-3.0.cl
def fixed_from_julian(j_date):
    """Return the fixed date equivalent to the Julian date 'j_date'."""
    month = standard_month(j_date)
    day = standard_day(j_date)
    year = standard_year(j_date)
    y = year + 1 if year < 0 else year
    return (JULIAN_EPOCH - 1 +
            (365 * (y - 1)) +
            quotient(y - 1, 4) +
            quotient(367*month - 362, 12) +
            (0 if month <= 2 else (-1 if is_julian_leap_year(year) else -2)) +
            day)

# see lines 1084-1111 in calendrica-3.0.cl
def julian_from_fixed(date):
    """Return the Julian date corresponding to fixed date 'date'."""
    approx = quotient(((4 * (date - JULIAN_EPOCH))) + 1464, 1461)
    year = approx - 1 if approx <= 0 else approx
    prior_days = date - fixed_from_julian(julian_date(year, JANUARY, 1))
    correction = (0 if date < fixed_from_julian(julian_date(year, MARCH, 1))
                  else (1 if is_julian_leap_year(year) else 2))
    month = quotient(12*(prior_days + correction) + 373, 367)
    day = 1 + (date - fixed_from_julian(julian_date(year, month, 1)))
    return julian_date(year, month, day)
```

This definition is continued in chunks 61 and 63.

This code is used in chunk 3.

Defines:

bce, used in chunks 61, 80, and 86.

ce, used in chunks 71, 77, 112, and 126.
 fixed_from_julian, used in chunks 60, 61, 63, 71, 74, 77, 80, 83, 86, 112, and 148.
 is_julian_leap_year, used in chunks 60 and 61.
 julian_date, used in chunks 60, 61, 63, 71, 74, 77, 80, 83, 86, 112, 139, and 164.
 JULIAN_EPOCH, never used.
 julian_from_fixed, used in chunks 60, 61, 63, and 148.
 Uses DECEMBER 53, fixed_from_gregorian 55, gregorian_date 52, JANUARY 53, MARCH 53, mod 15,
 quotient 14 14, standard_day 39, standard_month 39, and standard_year 39.

```

60  (julian calendar unit test 60)≡
    class JulianSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                julian_from_fixed(self.testvalue), julian_date(1945, OCTOBER, 30))
            self.assertEqual(
                roman_from_fixed(self.testvalue),
                roman_date(1945, NOVEMBER, KALENDS, 3, is_julian_leap_year(1945)))

        def testConversionToFixed(self):
            self.assertEqual(self.testvalue,
                fixed_from_julian(julian_date(1945, OCTOBER, 30)))
            self.assertEqual(
                self.testvalue,
                fixed_from_roman(roman_date(1945, NOVEMBER, KALENDS, 3,
                    is_julian_leap_year(1945))))

        def testLeapYear(self):
            self.assertTrue(is_julian_leap_year(2000))
            self.assertTrue(is_julian_leap_year(1900))
  
```

This definition is continued in chunks 62, 142, and 149.

This code is used in chunks 4 and 64.

Defines:

JulianSmokeTestCase, never used.

Uses fixed_from_julian 59, fixed_from_roman 61, is_julian_leap_year 59, julian_date 59,
 julian_from_fixed 59, KALENDS 61, NOVEMBER 53, OCTOBER 53, roman_date 61, and roman_from_fixed
 61.

And here is the Roman representation of the Julian calendar:

```
61  (julian_calendar 59)+≡
    # see lines 1113-1116 in calendrica-3.0.cl
    KALENDS = 1

    # see lines 1118-1121 in calendrica-3.0.cl
    NONES = 2

    # see lines 1123-1126 in calendrica-3.0.cl
    IDES = 3

    # see lines 1128-1131 in calendrica-3.0.cl
    def roman_date(year, month, event, count, leap):
        """Return the Roman date data structure."""
        return [year, month, event, count, leap]

    # see lines 1133-1135 in calendrica-3.0.cl
    def roman_year(date):
        """Return the year of Roman date 'date'."""
        return date[0]

    # see lines 1137-1139 in calendrica-3.0.cl
    def roman_month(date):
        """Return the month of Roman date 'date'."""
        return date[1]

    # see lines 1141-1143 in calendrica-3.0.cl
    def roman_event(date):
        """Return the event of Roman date 'date'."""
        return date[2]

    # see lines 1145-1147 in calendrica-3.0.cl
    def roman_count(date):
        """Return the count of Roman date 'date'."""
        return date[3]

    # see lines 1149-1151 in calendrica-3.0.cl
    def roman_leap(date):
        """Return the leap indicator of Roman date 'date'."""
        return date[4]

    # see lines 1153-1158 in calendrica-3.0.cl
    def ides_of_month(month):
        """Return the date of the Ides in Roman month 'month'."""
        return 15 if month in [MARCH, MAY, JULY, OCTOBER] else 13

    # see lines 1160-1163 in calendrica-3.0.cl
    def nones_of_month(month):
        """Return the date of Nones in Roman month 'month'."""
        return ides_of_month(month) - 8

    # see lines 1165-1191 in calendrica-3.0.cl
    def fixed_from_roman(r_date):
        """Return the fixed date corresponding to Roman date 'r_date'."""
        leap = roman_leap(r_date)
        count = roman_count(r_date)
        event = roman_event(r_date)
        month = roman_month(r_date)
        year = roman_year(r_date)
        return ({KALENDS: fixed_from_julian(julian_date(year, month, 1)),
                NONES: fixed_from_julian(julian_date(year,
```

```

month,
nones_of_month(month))),
IDES:    fixed_from_julian(julian_date(year,
month,
ides_of_month(month)))

}[event] -
count +
(0 if (is_julian_leap_year(year) and
(month == MARCH) and
(event == KALENDS) and
(16 >= count >= 6))
else 1) +
(1 if leap else 0))

# see lines 1193-1229 in calendrica-3.0.c1
def roman_from_fixed(date):
    """Return the Roman name corresponding to fixed date 'date'."""
    j_date = julian_from_fixed(date)
    month = standard_month(j_date)
    day = standard_day(j_date)
    year = standard_year(j_date)
    month_prime = amod(1 + month, 12)
    year_prime = (year if month_prime <> 1
                  else (year + 1 if (year <> -1) else 1))
    kalends1 = fixed_from_roman(
        roman_date(year_prime, month_prime, KALENDS, 1, False))

    if day == 1:
        res = roman_date(year, month, KALENDS, 1, False)
    elif day <= nones_of_month(month):
        res = roman_date(year, month, NONES, nones_of_month(month)-day+1, False)
    elif day <= ides_of_month(month):
        res = roman_date(year, month, IDES, ides_of_month(month)-day+1, False)
    elif (month <> FEBRUARY) or not is_julian_leap_year(year):
        res = roman_date(year_prime,
                          month_prime,
                          KALENDS,
                          kalends1 - date + 1,
                          False)
    elif day < 25:
        res = roman_date(year, MARCH, KALENDS, 30 - day, False)
    else:
        res = roman_date(year, MARCH, KALENDS, 31 - day, day == 25)
    return res

# see lines 1231-1234 in calendrica-3.0.c1
YEAR_ROME_FOUNDED = bce(753)

# see lines 1236-1241 in calendrica-3.0.c1
def julian_year_from_auc_year(year):
    """Return the Julian year equivalent to AUC year 'year'."""
    return ((year + YEAR_ROME_FOUNDED - 1)
            if (1 <= year <= (year - YEAR_ROME_FOUNDED))
            else (year + YEAR_ROME_FOUNDED))

# see lines 1243-1248 in calendrica-3.0.c1
def auc_year_from_julian_year(year):
    """Return the AUC year equivalent to Julian year 'year'."""
    return ((year - YEAR_ROME_FOUNDED - 1)
            if (YEAR_ROME_FOUNDED <= year <= -1)

```

```
else (year - YEAR_ROME_FOUNDED))
```

This code is used in chunk 3.

Defines:

```
auc_year_from_julian_year, never used.
fixed_from_roman, used in chunks 60, 62, and 148.
IDES, never used.
ides_of_month, never used.
julian_year_from_auc_year, never used.
KALENDS, used in chunks 60 and 62.
NONES, never used.
nones_of_month, never used.
roman_count, never used.
roman_date, used in chunks 60, 62, and 139.
roman_event, never used.
roman_from_fixed, used in chunks 60, 62, and 148.
roman_leap, never used.
roman_month, never used.
roman_year, never used.
YEAR_ROME_FOUNDED, never used.
```

Uses amod 15, bce 59, FEBRUARY 53, fixed_from_julian 59, is_julian_leap_year 59, julian_date 59, julian_from_fixed 59, JULY 53, MARCH 53, MAY 53, OCTOBER 53, standard_day 39, standard_month 39, and standard_year 39.

```
62  <julian calendar unit test 60>+≡
    class RomanSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                roman_from_fixed(self.testvalue),
                roman_date(1945, NOVEMBER, KALENDS, 3, False))

        def testConversionToFixed(self):
            self.assertEqual(self.testvalue,
                fixed_from_roman(roman_date(1945,
                                                NOVEMBER,
                                                KALENDS,
                                                3,
                                                False)))
```

This code is used in chunks 4 and 64.

Defines:

```
RomanSmokeTestCase, never used.
```

Uses fixed_from_roman 61, KALENDS 61, NOVEMBER 53, roman_date 61, and roman_from_fixed 61.

Here are some other interesting helper functions:

```
63 <julian calendar 59>+=
# see lines 1250-1266 in calendrica-3.0.cl
def julian_in_gregorian(j_month, j_day, g_year):
    """Return the list of the fixed dates of Julian month 'j_month', day
    'j_day' that occur in Gregorian year 'g_year'."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(julian_from_fixed(jan1))
    y_prime = 1 if (y == -1) else (y + 1)
    date1 = fixed_from_julian(julian_date(y, j_month, j_day))
    date2 = fixed_from_julian(julian_date(y_prime, j_month, j_day))
    return list_range([date1, date2], gregorian_year_range(g_year))

# see lines 1268-1272 in calendrica-3.0.cl
def eastern_orthodox_christmas(g_year):
    """Return the list of zero or one fixed dates of Eastern Orthodox Christmas
    in Gregorian year 'g_year'."""
    return julian_in_gregorian(DECEMBER, 25, g_year)
```

This code is used in chunk 3.

Defines:

```
eastern_orthodox_christmas, never used.
julian_in_gregorian, never used.
```

Uses DECEMBER 53, fixed_from_julian 59, gregorian_new_year 56, gregorian_year_range 56,
julian_date 59, julian_from_fixed 59, list_range 47, and standard_year 39.

2.4.1 Unit tests

```
64 <julianCalendarUnitTest.py 64>≡
# <generated code warning 1>
<import for testing 6>
from appendixCUnitTest import AppendixCTable1TestCaseBase
<julian calendar unit test 60>
<execute tests 5>
```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139.

2.5 Egyptian/Armenian Calendars

```
65 <egyptian and armenian calendars 65>≡
#####
# egyptian and armenian calendars algorithms #
#####
# see lines 515-518 in calendrica-3.0.c1
def egyptian_date(year, month, day):
    """Return the Egyptian date data structure."""
    return [year, month, day]

# see lines 520-525 in calendrica-3.0.c1
EGYPTIAN_EPOCH = fixed_from_jd(1448638)

# see lines 527-536 in calendrica-3.0.c1
def fixed_from_egyptian(e_date):
    """Return the fixed date corresponding to Egyptian date 'e_date'."""
    month = standard_month(e_date)
    day = standard_day(e_date)
    year = standard_year(e_date)
    return EGYPTIAN_EPOCH + (365*(year - 1)) + (30*(month - 1)) + (day - 1)

# see lines 538-553 in calendrica-3.0.c1
def egyptian_from_fixed(date):
    """Return the Egyptian date corresponding to fixed date 'date'."""
    days = date - EGYPTIAN_EPOCH
    year = 1 + quotient(days, 365)
    month = 1 + quotient(mod(days, 365), 30)
    day = days - (365*(year - 1)) - (30*(month - 1)) + 1
    return egyptian_date(year, month, day)

# see lines 555-558 in calendrica-3.0.c1
def armenian_date(year, month, day):
    """Return the Armenian date data structure."""
    return [year, month, day]

# see lines 560-564 in calendrica-3.0.c1
ARMENIAN_EPOCH = rd(201443)

# see lines 566-575 in calendrica-3.0.c1
def fixed_from_armenian(a_date):
    """Return the fixed date corresponding to Armenian date 'a_date'."""
    month = standard_month(a_date)
    day = standard_day(a_date)
    year = standard_year(a_date)
    return (ARMENIAN_EPOCH +
            fixed_from_egyptian(egyptian_date(year, month, day)) -
            EGYPTIAN_EPOCH)

# see lines 577-581 in calendrica-3.0.c1
def armenian_from_fixed(date):
    """Return the Armenian date corresponding to fixed date 'date'."""
    return egyptian_from_fixed(date + (EGYPTIAN_EPOCH - ARMENIAN_EPOCH))
```

This code is used in chunk 3.

Defines:

- armenian_date, used in chunks 66 and 139.
- ARMENIAN_EPOCH, never used.
- armenian_from_fixed, used in chunks 66 and 151.
- egyptian_date, used in chunks 66 and 139.
- EGYPTIAN_EPOCH, never used.
- egyptian_from_fixed, used in chunks 66 and 150.

fixed_from_armenian, used in chunks 66 and 151.
fixed_from_egyptian, used in chunks 66 and 150.
Uses fixed_from_jd 48, mod 15, quotient 14 14, rd 37, standard_day 39, standard_month 39,
and standard_year 39.

and now the tests ...

```
66 <egyptian and armenian calendars unit test 66>≡
class EgyptianSmokeTestCase(unittest.TestCase):
    def setUp(self):
        self.testvalue = 710347
        self.aDate = egyptian_date(2694, 7, 10)

    def testConversionFromFixed(self):
        self.assertEqual(
            egyptian_from_fixed(self.testvalue), self.aDate)

    def testConversionToFixed(self):
        self.assertEqual(
            self.testvalue, fixed_from_egyptian(self.aDate))

#####
class ArmenianSmokeTestCase(unittest.TestCase):
    def setUp(self):
        self.testvalue = 710347
        self.aDate = armenian_date(1395, 4, 5)

    def testConversionFromFixed(self):
        self.assertEqual(
            armenian_from_fixed(self.testvalue), self.aDate)

    def testConversionToFixed(self):
        self.assertEqual(
            self.testvalue, fixed_from_armenian(self.aDate))
```

This definition is continued in chunk 152.

This code is used in chunks 4 and 67.

Defines:

ArmenianSmokeTestCase, never used.

EgyptianSmokeTestCase, never used.

Uses armenian_date 65, armenian_from_fixed 65, egyptian_date 65, egyptian_from_fixed 65,
fixed_from_armenian 65, and fixed_from_egyptian 65.

2.5.1 Unit tests

The tests for Egyptian and Armenian calendars are following the usual pattern as described in section 1.1.

```
67 <egyptianAndArmenianCalendarsUnitTest.py 67>≡
# <generated code warning 1>
<import for testing 6>
from appendixCUnitTest import AppendixCTable1TestCaseBase
<egyptian and armenian calendars unit test 66>
<execute tests 5>
```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139.

2.6 ISO Calendar

```
68 <iso calendar 68>≡
#####
# ISO calendar algorithms #
#####
# see lines 979-981 in calendrica-3.0.c1
def iso_date(year, week, day):
    """Return the ISO date data structure."""
    return [year, week, day]

# see lines 983-985 in calendrica-3.0.c1
def iso_week(date):
    """Return the week of ISO date 'date'."""
    return date[1]

# see lines 987-989 in calendrica-3.0.c1
def iso_day(date):
    """Return the day of ISO date 'date'."""
    return date[2]

# see lines 991-993 in calendrica-3.0.c1
def iso_year(date):
    """Return the year of ISO date 'date'."""
    return date[0]

# see lines 995-1005 in calendrica-3.0.c1
def fixed_from_iso(i_date):
    """Return the fixed date equivalent to ISO date 'i_date'."""
    week = iso_week(i_date)
    day = iso_day(i_date)
    year = iso_year(i_date)
    return nth_kday(week, SUNDAY, gregorian_date(year - 1, DECEMBER, 28)) + day

# see lines 1007-1022 in calendrica-3.0.c1
def iso_from_fixed(date):
    """Return the ISO date corresponding to the fixed date 'date'."""
    approx = gregorian_year_from_fixed(date - 3)
    year = (approx +
            1 if date >= fixed_from_iso(iso_date(approx + 1, 1, 1))
            else approx)
    week = 1 + quotient(date - fixed_from_iso(iso_date(year, 1, 1)), 7)
    day = amod(date - rd(0), 7)
    return iso_date(year, week, day)

# see lines 1024-1032 in calendrica-3.0.c1
def is_iso_long_year(i_year):
    """Return True if ISO year 'i_year' is a long (53-week) year."""
    jan1 = day_of_week_from_fixed(gregorian_new_year(i_year))
    dec31 = day_of_week_from_fixed(gregorian_year_end(i_year))
    return (jan1 == THURSDAY) or (dec31 == THURSDAY)
```

This code is used in chunk 3.

Defines:

- fixed_from_iso, used in chunks 69 and 147.
- is_iso_long_year, never used.
- iso_date, used in chunks 69 and 139.
- iso_day, never used.
- iso_from_fixed, used in chunks 69 and 147.
- iso_week, never used.
- iso_year, never used.

Uses `amod` 15, `day_of_week_from_fixed` 38, `DECEMBER` 53, `gregorian_date` 52, `gregorian_new_year` 56, `gregorian_year_end` 56, `gregorian_year_from_fixed` 55, `nth_kday` 56, `quotient` 14 14, `rd` 37, `SUNDAY` 38, and `THURSDAY` 38.

```
69  <iso_calendar_unit_test 69>≡
    class ISOSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347
            self.aDate = iso_date(1945, 46, 1)

        def testConversionFromFixed(self):
            self.assertEqual(
                iso_from_fixed(self.testvalue), self.aDate)

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_iso(self.aDate))
```

This definition is continued in chunk 146.

This code is used in chunks 4 and 70.

Defines:

`ISOSmokeTestCase`, never used.

Uses `fixed_from_iso` 68, `iso_date` 68, and `iso_from_fixed` 68.

2.6.1 Unit tests

```
70  <isoCalendarUnitTest.py 70>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable1TestCaseBase
    <iso_calendar_unit_test 69>
    <execute tests 5>
```

Root chunk (not used in this document).

Uses `AppendixCTable1TestCaseBase` 139.

2.7 Coptic and Ethiopic Calendars

```
71 <coptic and ethiopic calendars 71>≡
# see lines 1277-1279 in calendrica-3.0.cl
#####
# coptic and ethiopic calendars algorithms #
#####
def coptic_date(year, month, day):
    """Return the Coptic date data structure."""
    return [year, month, day]

# see lines 1281-1284 in calendrica-3.0.cl
COPTIC_EPOCH = fixed_from_julian(julian_date(ce(284), AUGUST, 29))

# see lines 1286-1289 in calendrica-3.0.cl
def is_coptic_leap_year(c_year):
    """Return True if Coptic year 'c_year' is a leap year
    in the Coptic calendar."""
    return mod(c_year, 4) == 3

# see lines 1291-1301 in calendrica-3.0.cl
def fixed_from_coptic(c_date):
    """Return the fixed date of Coptic date 'c_date'."""
    month = standard_month(c_date)
    day   = standard_day(c_date)
    year  = standard_year(c_date)
    return (COPTIC_EPOCH - 1 +
            365 * (year - 1) +
            quotient(year, 4) +
            30 * (month - 1) +
            day)

# see lines 1303-1318 in calendrica-3.0.cl
def coptic_from_fixed(date):
    """Return the Coptic date equivalent of fixed date 'date'."""
    year = quotient((4 * (date - COPTIC_EPOCH)) + 1463, 1461)
    month = 1 + quotient(date - fixed_from_coptic(coptic_date(year, 1, 1)), 30)
    day   = date + 1 - fixed_from_coptic(coptic_date(year, month, 1))
    return coptic_date(year, month, day)

# see lines 1320-1323 in calendrica-3.0.cl
def ethiopic_date(year, month, day):
    """Return the Ethiopic date data structure."""
    return [year, month, day]

# see lines 1325-1328 in calendrica-3.0.cl
ETHIOPIC_EPOCH = fixed_from_julian(julian_date(ce(8), AUGUST, 29))

# see lines 1330-1339 in calendrica-3.0.cl
def fixed_from_ethiopic(e_date):
    """Return the fixed date corresponding to Ethiopic date 'e_date'."""
    month = standard_month(e_date)
    day   = standard_day(e_date)
    year  = standard_year(e_date)
    return (ETHIOPIC_EPOCH +
            fixed_from_coptic(coptic_date(year, month, day)) - COPTIC_EPOCH)

# see lines 1341-1345 in calendrica-3.0.cl
def ethiopic_from_fixed(date):
    """Return the Ethiopic date equivalent of fixed date 'date'."""
    return coptic_from_fixed(date + (COPTIC_EPOCH - ETHIOPIC_EPOCH))
```

```

# see lines 1347-1360 in calendrica-3.0.cl
def coptic_in_gregorian(c_month, c_day, g_year):
    """Return the list of the fixed dates of Coptic month 'c_month',
    day 'c_day' that occur in Gregorian year 'g_year'."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(coptic_from_fixed(jan1))
    date1 = fixed_from_coptic(coptic_date(y, c_month, c_day))
    date2 = fixed_from_coptic(coptic_date(y+1, c_month, c_day))
    return list_range([date1, date2], gregorian_year_range(g_year))

# see lines 1362-1366 in calendrica-3.0.cl
def coptic_christmas(g_year):
    """Return the list of zero or one fixed dates of Coptic Christmas
    dates in Gregorian year 'g_year'."""
    return coptic_in_gregorian(4, 29, g_year)

```

This code is used in chunk 3.

Defines:

- `coptic_christmas`, never used.
- `coptic_date`, used in chunks 72 and 139.
- `COPTIC_EPOCH`, never used.
- `coptic_from_fixed`, used in chunks 72, 80, and 154.
- `coptic_in_gregorian`, used in chunk 80.
- `ethiopic_date`, used in chunks 72 and 155.
- `ETHIOPIC_EPOCH`, never used.
- `ethiopic_from_fixed`, used in chunks 72 and 157.
- `fixed_from_coptic`, used in chunks 72 and 154.
- `fixed_from_ethiopic`, used in chunks 72 and 157.
- `is_coptic_leap_year`, never used.

Uses AUGUST 53, ce 59, `fixed_from_julian` 59, `gregorian_new_year` 56, `gregorian_year_range` 56, `julian_date` 59, `list_range` 47, mod 15, quotient 14 14, `standard_day` 39, `standard_month` 39, and `standard_year` 39.

```

72  <coptic and ethiopic calendars unit test 72>≡
    class CopticSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                coptic_from_fixed(self.testvalue), coptic_date(1662, 3, 3))

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_coptic(coptic_date(1662, 3, 3)))

    class EthiopicSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                ethiopic_from_fixed(self.testvalue), ethiopic_date(1938, 3, 3))

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_ethiopic(ethiopic_date(1938, 3, 3)))

```

This definition is continued in chunks 153 and 156.

This code is used in chunks 4 and 73.

Defines:

CopticSmokeTestCase, never used.

EthiopicSmokeTestCase, never used.

Uses coptic_date 71, coptic_from_fixed 71, ethiopic_date 71, ethiopic_from_fixed 71, fixed_from_coptic 71, and fixed_from_ethiopic 71.

2.7.1 Unit tests

```

73  <copticAndEthiopicCalendarsUnitTest.py 73>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable1TestCaseBase
    from appendixCUnitTest import AppendixCTable2TestCaseBase
    <coptic and ethiopic calendars unit test 72>
    <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable1TestCaseBase 139 and AppendixCTable2TestCaseBase 155.

2.8 Ecclesiastical Calendars

```
74 <ecclesiastical calendars 74>≡
#####
# ecclesiastical calendars algorithms #
#####
# see lines 1371-1385 in calendrica-3.0.cl
def orthodox_easter(g_year):
    """Return fixed date of Orthodox Easter in Gregorian year g_year."""
    shifted_epact = mod(14 + 11 * mod(g_year, 19), 30)
    j_year        = g_year if g_year > 0 else g_year - 1
    paschal_moon  = fixed_from_julian(
        julian_date(j_year, APRIL, 19)) - shifted_epact
    return kday_after(SUNDAY, paschal_moon)

# see lines 76-91 in calendrica-3.0.errata.cl
def alt_orthodox_easter(g_year):
    """Return fixed date of Orthodox Easter in Gregorian year g_year.
    Alternative calculation."""
    paschal_moon = (354 * g_year +
        30 * quotient((7 * g_year) + 8, 19) +
        quotient(g_year, 4) -
        quotient(g_year, 19) -
        273 +
        GREGORIAN_EPOCH)
    return kday_after(SUNDAY, paschal_moon)

# see lines 1401-1426 in calendrica-3.0.cl
def easter(g_year):
    """Return fixed date of Easter in Gregorian year g_year."""
    century = quotient(g_year, 100) + 1
    shifted_epact = mod(14 +
        11 * mod(g_year, 19) -
        quotient(3 * century, 4) +
        quotient(5 + (8 * century), 25), 30)
    adjusted_epact = ((shifted_epact + 1)
        if ((shifted_epact == 0) or ((shifted_epact == 1) and
            (10 < mod(g_year, 19))))
        else shifted_epact)
    paschal_moon = (fixed_from_gregorian(gregorian_date(g_year, APRIL, 19)) -
        adjusted_epact)
    return kday_after(SUNDAY, paschal_moon)

# see lines 1429-1431 in calendrica-3.0.cl
def pentecost(g_year):
    """Return fixed date of Pentecost in Gregorian year g_year."""
    return easter(g_year) + 49
```

This code is used in chunk 3.

Defines:

- alt_orthodox_easter, used in chunk 168.
- easter, used in chunks 164 and 168.
- orthodox_easter, used in chunk 168.
- pentecost, never used.

Uses APRIL 53, fixed_from_gregorian 55, fixed_from_julian 59, gregorian_date 52,
GREGORIAN_EPOCH 52, julian_date 59, kday_after 56, mod 15, quotient 14 14, and SUNDAY 38.

75 \langle *ecclesiastical calendars unit test 75* $\rangle \equiv$

This definition is continued in chunk 167.
This code is used in chunks 4 and 76.

2.8.1 Unit tests

76 \langle *ecclesiasticalCalendarsUnitTest.py 76* $\rangle \equiv$
 # \langle *generated code warning 1* \rangle
 \langle *import for testing 6* \rangle
 from appendixCUnitTest import AppendixCTable3TestCaseBase
 \langle *ecclesiastical calendars unit test 75* \rangle
 \langle *execute tests 5* \rangle

Root chunk (not used in this document).
Uses AppendixCTable3TestCaseBase 164.

2.9 Islamic Calendar

```
77 <islamic calendar 77>≡
#####
# islamic calendar algorithms #
#####
# see lines 1436-1439 in calendrica-3.0.cl
def islamic_date(year, month, day):
    """Return an Islamic date data structure."""
    return [year, month, day]

# see lines 1441-1444 in calendrica-3.0.cl
ISLAMIC_EPOCH = fixed_from_julian(julian_date(ce(622), JULY, 16))

# see lines 1446-1449 in calendrica-3.0.cl
def is_islamic_leap_year(i_year):
    """Return True if i_year is an Islamic leap year."""
    return mod(14 + 11 * i_year, 30) < 11

# see lines 1451-1463 in calendrica-3.0.cl
def fixed_from_islamic(i_date):
    """Return fixed date equivalent to Islamic date i_date."""
    month = standard_month(i_date)
    day   = standard_day(i_date)
    year  = standard_year(i_date)
    return (ISLAMIC_EPOCH - 1 +
            (year - 1) * 354 +
            quotient(3 + 11 * year, 30) +
            29 * (month - 1) +
            quotient(month, 2) +
            day)

# see lines 1465-1483 in calendrica-3.0.cl
def islamic_from_fixed(date):
    """Return Islamic date (year month day) corresponding to fixed date date."""
    year      = quotient(30 * (date - ISLAMIC_EPOCH) + 10646, 10631)
    prior_days = date - fixed_from_islamic(islamic_date(year, 1, 1))
    month     = quotient(11 * prior_days + 330, 325)
    day       = date - fixed_from_islamic(islamic_date(year, month, 1)) + 1
    return islamic_date(year, month, day)

# see lines 1485-1501 in calendrica-3.0.cl
def islamic_in_gregorian(i_month, i_day, g_year):
    """Return list of the fixed dates of Islamic month i_month, day i_day that
    occur in Gregorian year g_year."""
    jan1 = gregorian_new_year(g_year)
    y     = standard_year(islamic_from_fixed(jan1))
    date1 = fixed_from_islamic(islamic_date(y, i_month, i_day))
    date2 = fixed_from_islamic(islamic_date(y + 1, i_month, i_day))
    date3 = fixed_from_islamic(islamic_date(y + 2, i_month, i_day))
    return list_range([date1, date2, date3], gregorian_year_range(g_year))

# see lines 1503-1507 in calendrica-3.0.cl
def mawlid_an_nabi(g_year):
    """Return list of fixed dates of Mawlid_an_Nabi occurring in Gregorian
    year g_year."""
    return islamic_in_gregorian(3, 12, g_year)
```

This code is used in chunk 3.

Defines:

`fixed_from_islamic`, used in chunks 78 and 159.
`is_islamic_leap_year`, never used.
`islamic_date`, used in chunks 78, 132, and 155.
`ISLAMIC_EPOCH`, used in chunk 132.
`islamic_from_fixed`, used in chunks 78 and 159.
`islamic_in_gregorian`, never used.
`mawlid_an_nabi`, never used.

Uses `ce` 59, `fixed_from_julian` 59, `gregorian_new_year` 56, `gregorian_year_range` 56, `julian_date` 59, `JULY` 53, `list_range` 47, `mod` 15, `quotient` 14 14, `standard_day` 39, `standard_month` 39, and `standard_year` 39.

```
78 <islamic calendar unit test 78>≡
    class IslamicSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                islamic_from_fixed(self.testvalue), islamic_date(1364, 12, 6))

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_islamic(islamic_date(1364, 12, 6)))
```

This definition is continued in chunk 158.

This code is used in chunks 4 and 79.

Defines:

`IslamicSmokeTestCase`, never used.

Uses `fixed_from_islamic` 77, `islamic_date` 77, and `islamic_from_fixed` 77.

2.9.1 Unit tests

```
79 <islamicCalendarUnitTest.py 79>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable2TestCaseBase
    <islamic calendar unit test 78>
    <execute tests 5>
```

Root chunk (not used in this document).

Uses `AppendixCTable2TestCaseBase` 155.

2.10 Hebrew Calendar

```
80 <hebrew calendar 80>≡
#####
# hebrew calendar algorithms #
#####
# see lines 1512-1514 in calendrica-3.0.cl
def hebrew_date(year, month, day):
    """Return an Hebrew date data structure."""
    return [year, month, day]

# see lines 1516-1519 in calendrica-3.0.cl
NISAN = 1

# see lines 1521-1524 in calendrica-3.0.cl
IYYAR = 2

# see lines 1526-1529 in calendrica-3.0.cl
SIVAN = 3

# see lines 1531-1534 in calendrica-3.0.cl
TAMMUZ = 4

# see lines 1536-1539 in calendrica-3.0.cl
AV = 5

# see lines 1541-1544 in calendrica-3.0.cl
ELUL = 6

# see lines 1546-1549 in calendrica-3.0.cl
TISHRI = 7

# see lines 1551-1554 in calendrica-3.0.cl
MARHESHVAN = 8

# see lines 1556-1559 in calendrica-3.0.cl
KISLEV = 9

# see lines 1561-1564 in calendrica-3.0.cl
TEVET = 10

# see lines 1566-1569 in calendrica-3.0.cl
SHEVAT = 11

# see lines 1571-1574 in calendrica-3.0.cl
ADAR = 12

# see lines 1576-1579 in calendrica-3.0.cl
ADARII = 13

# see lines 1581-1585 in calendrica-3.0.cl
HEBREW_EPOCH = fixed_from_julian(julian_date(bce(3761), OCTOBER, 7))

# see lines 1587-1590 in calendrica-3.0.cl
def is_hebrew_leap_year(h_year):
    """Return True if h_year is a leap year on Hebrew calendar."""
    return mod(7 * h_year + 1, 19) < 7

# see lines 1592-1597 in calendrica-3.0.cl
def last_month_of_hebrew_year(h_year):
    """Return last month of Hebrew year."""
```



```

        return ADARII if is_hebrew_leap_year(h_year) else ADAR

# see lines 1599-1603 in calendrica-3.0.c1
def is_hebrew_sabbatical_year(h_year):
    """Return True if h_year is a sabbatical year on the Hebrew calendar."""
    return mod(h_year, 7) == 0

# see lines 1605-1617 in calendrica-3.0.c1
def last_day_of_hebrew_month(h_month, h_year):
    """Return last day of month h_month in Hebrew year h_year."""
    if ((h_month in [IYYAR, TAMMUZ, ELUL, TEVET, ADARII])
        or ((h_month == ADAR) and (not is_hebrew_leap_year(h_year)))
        or ((h_month == MARHESHVAN) and (not is_long_marheshvan(h_year)))
        or ((h_month == KISLEV) and is_short_kislev(h_year))):
        return 29
    else:
        return 30

# see lines 1619-1634 in calendrica-3.0.c1
def molad(h_month, h_year):
    """Return moment of mean conjunction of h_month in Hebrew h_year."""
    y = (h_year + 1) if (h_month < TISHRI) else h_year
    months_elapsed = h_month - TISHRI + quotient(235 * y - 234, 19)
    return (HEBREW_EPOCH -
            876/25920 +
            months_elapsed * (29 + hr(12) + 793/25920))

# see lines 1636-1663 in calendrica-3.0.c1
def hebrew_calendar_elapsed_days(h_year):
    """Return number of days elapsed from the (Sunday) noon prior
    to the epoch of the Hebrew calendar to the mean
    conjunction (molad) of Tishri of Hebrew year h_year,
    or one day later."""
    months_elapsed = quotient(235 * h_year - 234, 19)
    parts_elapsed = 12084 + 13753 * months_elapsed
    days = 29 * months_elapsed + quotient(parts_elapsed, 25920)
    return (days + 1) if (mod(3 * (days + 1), 7) < 3) else days

# see lines 1665-1670 in calendrica-3.0.c1
def hebrew_new_year(h_year):
    """Return fixed date of Hebrew new year h_year."""
    return (HEBREW_EPOCH +
            hebrew_calendar_elapsed_days(h_year) +
            hebrew_year_length_correction(h_year))

# see lines 1672-1684 in calendrica-3.0.c1
def hebrew_year_length_correction(h_year):
    """Return delays to start of Hebrew year h_year to keep ordinary
    year in range 353-356 and leap year in range 383-386."""
    # I had a bug... h_year = 1 instead of h_year - 1!!!
    ny0 = hebrew_calendar_elapsed_days(h_year - 1)
    ny1 = hebrew_calendar_elapsed_days(h_year)
    ny2 = hebrew_calendar_elapsed_days(h_year + 1)
    if ((ny2 - ny1) == 356):
        return 2
    elif ((ny1 - ny0) == 382):
        return 1
    else:
        return 0

# see lines 1686-1690 in calendrica-3.0.c1

```

```

def days_in_hebrew_year(h_year):
    """Return number of days in Hebrew year h_year."""
    return hebrew_new_year(h_year + 1) - hebrew_new_year(h_year)

# see lines 1692-1695 in calendrica-3.0.c1
def is_long_marheshvan(h_year):
    """Return True if Marheshvan is long in Hebrew year h_year."""
    return days_in_hebrew_year(h_year) in [355, 385]

# see lines 1697-1700 in calendrica-3.0.c1
def is_short_kislev(h_year):
    """Return True if Kislev is short in Hebrew year h_year."""
    return days_in_hebrew_year(h_year) in [353, 383]

# see lines 1702-1721 in calendrica-3.0.c1
def fixed_from_hebrew(h_date):
    """Return fixed date of Hebrew date h_date."""
    month = standard_month(h_date)
    day = standard_day(h_date)
    year = standard_year(h_date)

    if (month < TISHRI):
        tmp = (summa(lambda m: last_day_of_hebrew_month(m, year),
                     TISHRI,
                     lambda m: m <= last_month_of_hebrew_year(year)) +
              summa(lambda m: last_day_of_hebrew_month(m, year),
                     NISAN,
                     lambda m: m < month))
    else:
        tmp = summa(lambda m: last_day_of_hebrew_month(m, year),
                     TISHRI,
                     lambda m: m < month)

    return hebrew_new_year(year) + day - 1 + tmp

# see lines 1723-1751 in calendrica-3.0.c1
def hebrew_from_fixed(date):
    """Return Hebrew (year month day) corresponding to fixed date date.
    # The fraction can be approximated by 365.25."""
    approx = quotient(date - HEBREW_EPOCH, 35975351/98496) + 1
    year = final(approx - 1, lambda y: hebrew_new_year(y) <= date)
    start = (TISHRI
             if (date < fixed_from_hebrew(hebrew_date(year, NISAN, 1)))
             else NISAN)
    month = next(start, lambda m: date <= fixed_from_hebrew(
        hebrew_date(year, m, last_day_of_hebrew_month(m, year))))
    day = date - fixed_from_hebrew(hebrew_date(year, month, 1)) + 1
    return hebrew_date(year, month, day)

# see lines 1753-1761 in calendrica-3.0.c1
def yom_kippur(g_year):
    """Return fixed date of Yom Kippur occurring in Gregorian year g_year."""
    hebrew_year = g_year - gregorian_year_from_fixed(HEBREW_EPOCH) + 1
    return fixed_from_hebrew(hebrew_date(hebrew_year, TISHRI, 10))

# see lines 1763-1770 in calendrica-3.0.c1
def passover(g_year):
    """Return fixed date of Passover occurring in Gregorian year g_year."""
    hebrew_year = g_year - gregorian_year_from_fixed(HEBREW_EPOCH)
    return fixed_from_hebrew(hebrew_date(hebrew_year, NISAN, 15))

```

```

# see lines 1772-1782 in calendrica-3.0.cl
def omer(date):
    """Return the number of elapsed weeks and days in the omer at date date.
    Returns BOGUS if that date does not fall during the omer."""
    c = date - passover(gregorian_year_from_fixed(date))
    return [quotient(c, 7), mod(c, 7)] if (1 <= c <= 49) else BOGUS

# see lines 1784-1793 in calendrica-3.0.cl
def purim(g_year):
    """Return fixed date of Purim occurring in Gregorian year g_year."""
    hebrew_year = g_year - gregorian_year_from_fixed(HEBREW_EPOCH)
    last_month = last_month_of_hebrew_year(hebrew_year)
    return fixed_from_hebrew(hebrew_date(hebrew_year(last_month, 14)))

# see lines 1795-1805 in calendrica-3.0.cl
def ta_anit_esther(g_year):
    """Return fixed date of Ta'anit Esther occurring in Gregorian
    year g_year."""
    purim_date = purim(g_year)
    return ((purim_date - 3)
            if (day_of_week_from_fixed(purim_date) == SUNDAY)
            else (purim_date - 1))

# see lines 1807-1821 in calendrica-3.0.cl
def tishah_be_av(g_year):
    """Return fixed date of Tishah be'Av occurring in Gregorian year g_year."""
    hebrew_year = g_year - gregorian_year_from_fixed(HEBREW_EPOCH)
    av9 = fixed_from_hebrew(hebrew_date(hebrew_year, AV, 9))
    return (av9 + 1) if (day_of_week_from_fixed(av9) == SATURDAY) else av9

# see lines 1823-1834 in calendrica-3.0.cl
def birkath_ha_hama(g_year):
    """Return the list of fixed date of Birkath ha'Hama occurring in
    Gregorian year g_year, if it occurs."""
    dates = coptic_in_gregorian(7, 30, g_year)
    return (dates
            if ((not (dates == [])) and
                (mod(standard_year(coptic_from_fixed(dates[0])), 28) == 17))
            else [])

# see lines 1836-1840 in calendrica-3.0.cl
def sh_ela(g_year):
    """Return the list of fixed dates of Sh'ela occurring in
    Gregorian year g_year."""
    return coptic_in_gregorian(3, 26, g_year)

# exercise for the reader from pag 104
def hebrew_in_gregorian(h_month, h_day, g_year):
    """Return list of the fixed dates of Hebrew month, h_month, day, h_day,
    that occur in Gregorian year g_year."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(hebrew_from_fixed(jan1))
    date1 = fixed_from_hebrew(hebrew_date(y, h_month, h_day))
    date2 = fixed_from_hebrew(hebrew_date(y + 1, h_month, h_day))
    # Hebrew and Gregorian calendar are aligned but certain
    # holidays, i.e. Tzom Tevet, can fall on either side of Jan 1.
    # So we can have 0, 1 or 2 occurrences of that holiday.
    dates = [date1, date2]
    return list_range(dates, gregorian_year_range(g_year))

# see pag 104

```

```

def tzom_tevet(g_year):
    """Return the list of fixed dates for Tzom Tevet (Tevet 10) that
    occur in Gregorian year g_year. It can occur 0, 1 or 2 times per
    Gregorian year."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(hebrew_from_fixed(jan1))
    d1 = fixed_from_hebrew(hebrew_date(y, TEVET, 10))
    d1 = (d1 + 1) if (day_of_week_from_fixed(d1) == SATURDAY) else d1
    d2 = fixed_from_hebrew(hebrew_date(y + 1, TEVET, 10))
    d2 = (d2 + 1) if (day_of_week_from_fixed(d2) == SATURDAY) else d2
    dates = [d1, d2]
    return list_range(dates, gregorian_year_range(g_year))

# this is a simplified version where no check for SATURDAY
# is performed: from hebrew year 1 till 2000000
# there is no TEVET 10 falling on Saturday...
def alt_tzom_tevet(g_year):
    """Return the list of fixed dates for Tzom Tevet (Tevet 10) that
    occur in Gregorian year g_year. It can occur 0, 1 or 2 times per
    Gregorian year."""
    return hebrew_in_gregorian(TEVET, 10, g_year)

# see lines 1842-1859 in calendrica-3.0.c1
def yom_ha_zikkaron(g_year):
    """Return fixed date of Yom ha_Zikkaron occurring in Gregorian
    year g_year."""
    hebrew_year = g_year - gregorian_year_from_fixed(HEBREW_EPOCH)
    iyyar4 = fixed_from_hebrew(hebrew_date(hebrew_year, IYYAR, 4))

    if (day_of_week_from_fixed(iyyar4) in [THURSDAY, FRIDAY]):
        return kday_before(WEDNESDAY, iyyar4)
    elif (SUNDAY == day_of_week_from_fixed(iyyar4)):
        return iyyar4 + 1
    else:
        return iyyar4

# see lines 1861-1879 in calendrica-3.0.c1
def hebrew_birthday(birthdate, h_year):
    """Return fixed date of the anniversary of Hebrew birth date
    birthdate occurring in Hebrew h_year."""
    birth_day = standard_day(birthdate)
    birth_month = standard_month(birthdate)
    birth_year = standard_year(birthdate)
    if (birth_month == last_month_of_hebrew_year(birth_year)):
        return fixed_from_hebrew(hebrew_date(h_year,
                                              last_month_of_hebrew_year(h_year),
                                              birth_day))
    else:
        return (fixed_from_hebrew(hebrew_date(h_year, birth_month, 1)) +
                birth_day - 1)

# see lines 1881-1893 in calendrica-3.0.c1
def hebrew_birthday_in_gregorian(birthdate, g_year):
    """Return the list of the fixed dates of Hebrew birthday
    birthday that occur in Gregorian g_year."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(hebrew_from_fixed(jan1))
    date1 = hebrew_birthday(birthdate, y)
    date2 = hebrew_birthday(birthdate, y + 1)
    return list_range([date1, date2], gregorian_year_range(g_year))

```

```

# see lines 1895-1937 in calendrica-3.0.c1
def yahrzeit(death_date, h_year):
    """Return fixed date of the anniversary of Hebrew death date death_date
    occurring in Hebrew h_year."""
    death_day = standard_day(death_date)
    death_month = standard_month(death_date)
    death_year = standard_year(death_date)

    if ((death_month == MARHESHVAN) and
        (death_day == 30) and
        (not is_long_marheshvan(death_year + 1))):
        return fixed_from_hebrew(hebrew_date(h_year, KISLEV, 1)) - 1
    elif ((death_month == KISLEV) and
          (death_day == 30) and
          is_short_kislev(death_year + 1)):
        return fixed_from_hebrew(hebrew_date(h_year, TEVET, 1)) - 1
    elif (death_month == ADARII):
        return fixed_from_hebrew(hebrew_date(h_year,
                                                last_month_of_hebrew_year(h_year),
                                                death_day))

    elif ((death_day == 30) and
          (death_month == ADAR) and
          (not is_hebrew_leap_year(h_year))):
        return fixed_from_hebrew(hebrew_date(h_year, SHEVAT, 30))
    else:
        return (fixed_from_hebrew(hebrew_date(h_year, death_month, 1)) +
                death_day - 1)

# see lines 1939-1951 in calendrica-3.0.c1
def yahrzeit_in_gregorian(death_date, g_year):
    """Return the list of the fixed dates of death date death_date (yahrzeit)
    that occur in Gregorian year g_year."""
    jan1 = gregorian_new_year(g_year)
    y = standard_year(hebrew_from_fixed(jan1))
    date1 = yahrzeit(death_date, y)
    date2 = yahrzeit(death_date, y + 1)
    return list_range([date1, date2], gregorian_year_range(g_year))

# see lines 1953-1960 in calendrica-3.0.c1
def shift_days(l, cap_Delta):
    """Shift each weekday on list l by cap_Delta days."""
    return map(lambda x: day_of_week_from_fixed(x + cap_Delta), l)

# see lines 1962-1984 in calendrica-3.0.c1
def possible_hebrew_days(h_month, h_day):
    """Return a list of possible days of week for Hebrew day h_day
    and Hebrew month h_month."""
    h_date0 = hebrew_date(5, NISAN, 1)
    h_year = 6 if (h_month > ELUL) else 5
    h_date = hebrew_date(h_year, h_month, h_day)
    n = fixed_from_hebrew(h_date) - fixed_from_hebrew(h_date0)
    tue_thu_sat = [TUESDAY, THURSDAY, SATURDAY]

    if (h_day == 30) and (h_month in [MARHESHVAN, KISLEV]):
        sun_wed_fri = []
    elif (h_month == KISLEV):
        sun_wed_fri = [SUNDAY, WEDNESDAY, FRIDAY]
    else:
        sun_wed_fri = [SUNDAY]

    mon = [MONDAY] if h_month in [KISLEV, TEVET, SHEVAT, ADAR] else [ ]

```

```

    ell = tue_thu_sat
    ell.extend(sun_wed_fri)
    ell.extend(mon)
    return shift_days(ell, n)

```

This code is used in chunk 3.

Defines:

```

ADAR, never used.
ADARII, never used.
alt_tzom_tevet, never used.
AV, never used.
birkath_ha_hama, used in chunk 81.
days_in_hebrew_year, never used.
ELUL, never used.
fixed_from_hebrew, used in chunks 81, 132, and 166.
hebrew_birthday, never used.
hebrew_birthday_in_gregorian, never used.
hebrew_calendar_elapsed_days, never used.
hebrew_date, used in chunks 81, 132, 164, and 243.
HEBREW_EPOCH, never used.
hebrew_from_fixed, used in chunks 81, 132, and 166.
hebrew_in_gregorian, never used.
hebrew_new_year, never used.
hebrew_year_length_correction, never used.
is_hebrew_leap_year, never used.
is_hebrew_sabbatical_year, never used.
is_long_marheshvan, never used.
is_short_kislev, never used.
IYYAR, never used.
KISLEV, used in chunk 81.
last_day_of_hebrew_month, never used.
last_month_of_hebrew_year, never used.
MARHESHVAN, never used.
molad, never used.
NISAN, used in chunk 132.
omer, never used.
passover, never used.
possible_hebrew_days, used in chunk 81.
purim, never used.
sh_eia, never used.
SHEVAT, used in chunk 81.
shift_days, never used.
SIVAN, never used.
ta_anit_esther, never used.
TAMMUZ, never used.
TEVET, never used.
tishah_be_av, never used.
TISHRI, used in chunk 132.
tzom_tevet, used in chunk 81.
yahrzeit, never used.
yahrzeit_in_gregorian, never used.
yom_ha_zikkaron, never used.
yom_kippur, never used.

```

Uses bce 59, BOGUS 13, coptic_from_fixed 71, coptic_in_gregorian 71, day_of_week_from_fixed 38, epoch 37, final 19, fixed_from_julian 59, FRIDAY 38, gregorian_new_year 56, gregorian_year_from_fixed 55, gregorian_year_range 56, hr 92, julian_date 59, kday_before 56, list_range 47, mod 15, MONDAY 38, next 16, OCTOBER 53, quotient 14 14, SATURDAY 38, standard_day 39, standard_month 39, standard_year 39, start 47, summa 22, SUNDAY 38, THURSDAY 38, TUESDAY 38, and WEDNESDAY 38.

```

81  <hebrew calendar unit test 81>≡
    class HebrewSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                hebrew_from_fixed(self.testvalue), hebrew_date(5706, KISLEV, 7))

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue, fixed_from_hebrew(hebrew_date(5706, KISLEV, 7)))

    class HebrewHolidaysTestCase(unittest.TestCase):
        def testBirkathHaHama(self):
            self.assertNotEqual(birkath_ha_hama(1925), [])
            self.assertEqual(birkath_ha_hama(1926), [])
            self.assertNotEqual(birkath_ha_hama(1925+28), [])

        def testTzomTevet(self):
            """test tzom tevet (Tevet 10): see page 104"""
            self.assertEqual(len(tzom_tevet(1982)), 2)
            self.assertEqual(len(tzom_tevet(1984)), 0)

        def testPossibleHebrewDays(self):
            """see page 110, Calendrical Calculations, 3rd edition."""
            self.assertEqual(set(possible_hebrew_days(SHEVAT, 15)),
                set([THURSDAY, SATURDAY, MONDAY, TUESDAY, WEDNESDAY]))

```

This definition is continued in chunk 165.

This code is used in chunks 4 and 82.

Defines:

HebrewHolidaysTestCase, never used.
 HebrewSmokeTestCase, never used.
 testBirkathHaHama, never used.
 testPossibleHebrewDays, never used.
 testTzomTevet, never used.

Uses birkath_ha_hama 80, fixed_from_hebrew 80, hebrew_date 80, hebrew_from_fixed 80, KISLEV 80,
 MONDAY 38, possible_hebrew_days 80, SATURDAY 38, SHEVAT 80, THURSDAY 38, TUESDAY 38,
 tzom_tevet 80, and WEDNESDAY 38.

2.10.1 Unit tests

```

82  <hebrewCalendarUnitTest.py 82>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable3TestCaseBase
    <hebrew calendar unit test 81>
    <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable3TestCaseBase 164.

2.11 Mayan Calendars

```
83  <mayan calendars 83>≡
#####
# mayan calendars algorithms #
#####
# see lines 1989-1992 in calendrica-3.0.cl
def mayan_long_count_date(baktun, katun, tun, uinal, kin):
    """Return a long count Mayan date data structure."""
    return [baktun, katun, tun, uinal, kin]

# see lines 1994-1996 in calendrica-3.0.cl
def mayan_haab_date(month, day):
    """Return a Haab Mayan date data structure."""
    return [month, day]

# see lines 1998-2001 in calendrica-3.0.cl
def mayan_tzolkin_date(number, name):
    """Return a Tzolkin Mayan date data structure."""
    return [number, name]

# see lines 2003-2005 in calendrica-3.0.cl
def mayan_baktun(date):
    """Return the baktun field of a long count Mayan
    date = [baktun, katun, tun, uinal, kin]."""
    return date[0]

# see lines 2007-2009 in calendrica-3.0.cl
def mayan_katun(date):
    """Return the katun field of a long count Mayan
    date = [baktun, katun, tun, uinal, kin]."""
    return date[1]

# see lines 2011-2013 in calendrica-3.0.cl
def mayan_tun(date):
    """Return the tun field of a long count Mayan
    date = [baktun, katun, tun, uinal, kin]."""
    return date[2]

# see lines 2015-2017 in calendrica-3.0.cl
def mayan_uinal(date):
    """Return the uinal field of a long count Mayan
    date = [baktun, katun, tun, uinal, kin]."""
    return date[3]

# see lines 2019-2021 in calendrica-3.0.cl
def mayan_kin(date):
    """Return the kin field of a long count Mayan
    date = [baktun, katun, tun, uinal, kin]."""
    return date[4]

# see lines 2023-2025 in calendrica-3.0.cl
def mayan_haab_month(date):
    """Return the month field of Haab Mayan date = [month, day]."""
    return date[0]

# see lines 2027-2029 in calendrica-3.0.cl
def mayan_haab_day(date):
    """Return the day field of Haab Mayan date = [month, day]."""
    return date[1]
```



```

# see lines 2031-2033 in calendrica-3.0.cl
def mayan_tzolkin_number(date):
    """Return the number field of Tzolkin Mayan date = [number, name]."""
    return date[0]

# see lines 2035-2037 in calendrica-3.0.cl
def mayan_tzolkin_name(date):
    """Return the name field of Tzolkin Mayan date = [number, name]."""
    return date[1]

# see lines 2039-2044 in calendrica-3.0.cl
MAYAN_EPOCH = fixed_from_jd(584283)

# see lines 2046-2060 in calendrica-3.0.cl
def fixed_from_mayan_long_count(count):
    """Return fixed date corresponding to the Mayan long count count,
    which is a list [baktun, katun, tun, uinal, kin]."""
    baktun = mayan_baktun(count)
    katun = mayan_katun(count)
    tun = mayan_tun(count)
    uinal = mayan_uinal(count)
    kin = mayan_kin(count)
    return (MAYAN_EPOCH +
            (baktun * 144000) +
            (katun * 7200) +
            (tun * 360) +
            (uinal * 20) +
            kin)

# see lines 2062-2074 in calendrica-3.0.cl
def mayan_long_count_from_fixed(date):
    """Return Mayan long count date of fixed date date."""
    long_count = date - MAYAN_EPOCH
    baktun, day_of_baktun = divmod(long_count, 144000)
    katun, day_of_katun = divmod(day_of_baktun, 7200)
    tun, day_of_tun = divmod(day_of_katun, 360)
    uinal, kin = divmod(day_of_tun, 20)
    return mayan_long_count_date(baktun, katun, tun, uinal, kin)

# see lines 2076-2081 in calendrica-3.0.cl
def mayan_haab_ordinal(h_date):
    """Return the number of days into cycle of Mayan haab date h_date."""
    day = mayan_haab_day(h_date)
    month = mayan_haab_month(h_date)
    return ((month - 1) * 20) + day

# see lines 2083-2087 in calendrica-3.0.cl
MAYAN_HAAB_EPOCH = MAYAN_EPOCH - mayan_haab_ordinal(mayan_haab_date(18, 8))

# see lines 2089-2096 in calendrica-3.0.cl
def mayan_haab_from_fixed(date):
    """Return Mayan haab date of fixed date date."""
    count = mod(date - MAYAN_HAAB_EPOCH, 365)
    day = mod(count, 20)
    month = quotient(count, 20) + 1
    return mayan_haab_date(month, day)

# see lines 2098-2105 in calendrica-3.0.cl
def mayan_haab_on_or_before(haab, date):
    """Return fixed date of latest date on or before fixed date date
    that is Mayan haab date haab."""

```

```

    return date - mod(date - MAYAN_HAAB_EPOCH - mayan_haab_ordinal(haab), 365)

# see lines 2107-2114 in calendrica-3.0.c1
def mayan_tzolkin_ordinal(t_date):
    """Return number of days into Mayan tzolkin cycle of t_date."""
    number = mayan_tzolkin_number(t_date)
    name = mayan_tzolkin_name(t_date)
    return mod(number - 1 + (39 * (number - name)), 260)

# see lines 2116-2120 in calendrica-3.0.c1
MAYAN_TZOLKIN_EPOCH = (MAYAN_EPOCH -
    mayan_tzolkin_ordinal(mayan_tzolkin_date(4, 20)))

# see lines 2122-2128 in calendrica-3.0.c1
def mayan_tzolkin_from_fixed(date):
    """Return Mayan tzolkin date of fixed date date."""
    count = date - MAYAN_TZOLKIN_EPOCH + 1
    number = amod(count, 13)
    name = amod(count, 20)
    return mayan_tzolkin_date(number, name)

# see lines 2130-2138 in calendrica-3.0.c1
def mayan_tzolkin_on_or_before(tzolkin, date):
    """Return fixed date of latest date on or before fixed date date
    that is Mayan tzolkin date tzolkin."""
    return (date -
        mod(date -
            MAYAN_TZOLKIN_EPOCH -
            mayan_tzolkin_ordinal(tzolkin), 260))

# see lines 2140-2150 in calendrica-3.0.c1
def mayan_year_bearer_from_fixed(date):
    """Return year bearer of year containing fixed date date.
    Returns BOGUS for uayeb."""
    x = mayan_haab_on_or_before(mayan_haab_date(1, 0), date + 364)
    return (BOGUS if (mayan_haab_month(mayan_haab_from_fixed(date)) == 19)
        else mayan_tzolkin_name(mayan_tzolkin_from_fixed(x)))

# see lines 2152-2168 in calendrica-3.0.c1
def mayan_calendar_round_on_or_before(haab, tzolkin, date):
    """Return fixed date of latest date on or before date, that is
    Mayan haab date haab and tzolkin date tzolkin.
    Returns BOGUS for impossible combinations."""
    haab_count = mayan_haab_ordinal(haab) + MAYAN_HAAB_EPOCH
    tzolkin_count = mayan_tzolkin_ordinal(tzolkin) + MAYAN_TZOLKIN_EPOCH
    diff = tzolkin_count - haab_count
    if mod(diff, 5) == 0:
        return date - mod(date - haab_count(365 * diff), 18980)
    else:
        return BOGUS

# see lines 2170-2173 in calendrica-3.0.c1
def aztec_xihuitl_date(month, day):
    """Return an Aztec xihuitl date data structure."""
    return [month, day]

# see lines 2175-2177 in calendrica-3.0.c1
def aztec_xihuitl_month(date):
    """Return the month field of an Aztec xihuitl date = [month, day]."""
    return date[0]

```

```

# see lines 2179-2181 in calendrica-3.0.c1
def aztec_xihuitl_day(date):
    """Return the day field of an Aztec xihuitl date = [month, day]."""
    return date[1]

# see lines 2183-2186 in calendrica-3.0.c1
def aztec_tonalpohualli_date(number, name):
    """Return an Aztec tonalpohualli date data structure."""
    return [number, name]

# see lines 2188-2191 in calendrica-3.0.c1
def aztec_tonalpohualli_number(date):
    """Return the number field of an Aztec tonalpohualli
    date = [number, name]."""
    return date[0]

# see lines 2193-2195 in calendrica-3.0.c1
def aztec_tonalpohualli_name(date):
    """Return the name field of an Aztec tonalpohualli
    date = [number, name]."""
    return date[1]

# see lines 2197-2200 in calendrica-3.0.c1
def aztec_xiuhmolpilli_designation(number, name):
    """Return an Aztec xiuhmolpilli date data structure."""
    return [number, name]

# see lines 2202-2205 in calendrica-3.0.c1
def aztec_xiuhmolpilli_number(date):
    """Return the number field of an Aztec xiuhmolpilli
    date = [number, name]."""
    return date[0]

# see lines 2207-2210 in calendrica-3.0.c1
def aztec_xiuhmolpilli_name(date):
    """Return the name field of an Aztec xiuhmolpilli
    date = [number, name]."""
    return date[1]

# see lines 2212-2215 in calendrica-3.0.c1
AZTEC_CORRELATION = fixed_from_julian(julian_date(1521, AUGUST, 13))

# see lines 2217-2223 in calendrica-3.0.c1
def aztec_xihuitl_ordinal(x_date):
    """Return the number of elapsed days into cycle of Aztec xihuitl
    date x_date."""
    day = aztec_xihuitl_day(x_date)
    month = aztec_xihuitl_month(x_date)
    return ((month - 1) * 20) + day - 1

# see lines 2225-2229 in calendrica-3.0.c1
AZTEC_XIHUITL_CORRELATION = (AZTEC_CORRELATION -
    aztec_xihuitl_ordinal(aztec_xihuitl_date(11, 2)))

# see lines 2231-2237 in calendrica-3.0.c1
def aztec_xihuitl_from_fixed(date):
    """Return Aztec xihuitl date of fixed date date."""
    count = mod(date - AZTEC_XIHUITL_CORRELATION, 365)
    day = mod(count, 20) + 1
    month = quotient(count, 20) + 1

```

```

    return aztec_xihuitl_date(month, day)

# see lines 2239-2246 in calendrica-3.0.cl
def aztec_xihuitl_on_or_before(xihuitl, date):
    """Return fixed date of latest date on or before fixed date date
    that is Aztec xihuitl date xihuitl."""
    return (date -
            mod(date -
                AZTEC_XIHUITL_CORRELATION -
                aztec_xihuitl_ordinal(xihuitl), 365))

# see lines 2248-2255 in calendrica-3.0.cl
def aztec_tonalpohualli_ordinal(t_date):
    """Return the number of days into Aztec tonalpohualli cycle of t_date."""
    number = aztec_tonalpohualli_number(t_date)
    name = aztec_tonalpohualli_name(t_date)
    return mod(number - 1 + 39 * (number - name), 260)

# see lines 2257-2262 in calendrica-3.0.cl
AZTEC_TONALPOHUALLI_CORRELATION = (AZTEC_CORRELATION -
                                    aztec_tonalpohualli_ordinal(
                                        aztec_tonalpohualli_date(1, 5)))

# see lines 2264-2270 in calendrica-3.0.cl
def aztec_tonalpohualli_from_fixed(date):
    """Return Aztec tonalpohualli date of fixed date date."""
    count = date - AZTEC_TONALPOHUALLI_CORRELATION + 1
    number = amod(count, 13)
    name = amod(count, 20)
    return aztec_tonalpohualli_date(number, name)

# see lines 2272-2280 in calendrica-3.0.cl
def aztec_tonalpohualli_on_or_before(tonalpohualli, date):
    """Return fixed date of latest date on or before fixed date date
    that is Aztec tonalpohualli date tonalpohualli."""
    return (date -
            mod(date -
                AZTEC_TONALPOHUALLI_CORRELATION -
                aztec_tonalpohualli_ordinal(tonalpohualli), 260))

# see lines 2282-2303 in calendrica-3.0.cl
def aztec_xihuitl_tonalpohualli_on_or_before(xihuitl, tonalpohualli, date):
    """Return fixed date of latest xihuitl_tonalpohualli combination
    on or before date date. That is the date on or before
    date date that is Aztec xihuitl date xihuitl and
    tonalpohualli date tonalpohualli.
    Returns BOGUS for impossible combinations."""
    xihuitl_count = aztec_xihuitl_ordinal(xihuitl) + AZTEC_XIHUITL_CORRELATION
    tonalpohualli_count = (aztec_tonalpohualli_ordinal(tonalpohualli) +
                           AZTEC_TONALPOHUALLI_CORRELATION)
    diff = tonalpohualli_count - xihuitl_count
    if mod(diff, 5) == 0:
        return date - mod(date - xihuitl_count - (365 * diff), 18980)
    else:
        return BOGUS

# see lines 2305-2316 in calendrica-3.0.cl
def aztec_xiuhmolpilli_from_fixed(date):
    """Return designation of year containing fixed date date.
    Returns BOGUS for nemontemi."""
    x = aztec_xihuitl_on_or_before(aztec_xihuitl_date(18, 20), date + 364)

```

```

month = aztec_xihuitl_month(aztec_xihuitl_from_fixed(date))
return BOGUS if (month == 19) else aztec_tonalpohualli_from_fixed(x)

```

This code is used in chunk 3.

Defines:

```

AZTEC_CORRELATION, never used.
AZTEC_TONALPOHUALLI_CORRELATION, never used.
aztec_tonalpohualli_date, used in chunk 155.
aztec_tonalpohualli_from_fixed, used in chunk 163.
aztec_tonalpohualli_name, never used.
aztec_tonalpohualli_number, never used.
aztec_tonalpohualli_on_or_before, never used.
aztec_tonalpohualli_ordinal, never used.
AZTEC_XIHUITL_CORRELATION, never used.
aztec_xihuitl_date, used in chunks 84 and 155.
aztec_xihuitl_day, never used.
aztec_xihuitl_from_fixed, used in chunks 84 and 163.
aztec_xihuitl_month, never used.
aztec_xihuitl_on_or_before, used in chunk 84.
aztec_xihuitl_ordinal, never used.
aztec_xihuitl_tonalpohualli_on_or_before, never used.
aztec_xiuhmolpilli_designation, never used.
aztec_xiuhmolpilli_from_fixed, never used.
aztec_xiuhmolpilli_name, never used.
aztec_xiuhmolpilli_number, never used.
fixed_from_mayan_long_count, used in chunks 84 and 163.
mayan_baktun, never used.
mayan_calendar_round_on_or_before, never used.
MAYAN_EPOCH, never used.
mayan_haab_date, used in chunks 84 and 155.
mayan_haab_day, never used.
MAYAN_HAAB_EPOCH, never used.
mayan_haab_from_fixed, used in chunks 84 and 163.
mayan_haab_month, never used.
mayan_haab_on_or_before, used in chunk 84.
mayan_haab_ordinal, never used.
mayan_katun, never used.
mayan_kin, never used.
mayan_long_count_date, used in chunks 84 and 155.
mayan_long_count_from_fixed, used in chunks 84 and 163.
mayan_tun, never used.
mayan_tzolkin_date, used in chunks 84 and 155.
MAYAN_TZOLKIN_EPOCH, never used.
mayan_tzolkin_from_fixed, used in chunks 84 and 163.
mayan_tzolkin_name, never used.
mayan_tzolkin_number, never used.
mayan_tzolkin_on_or_before, used in chunk 84.
mayan_tzolkin_ordinal, never used.
mayan_uinal, never used.
mayan_year_bearer_from_fixed, never used.

```

Uses amod 15, AUGUST 53, BOGUS 13, fixed_from_jd 48, fixed_from_julian 59, julian_date 59, mod 15, and quotient 14 14.

```

84  <mayan calendars unit test 84>≡
    class MayanSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(mayan_long_count_from_fixed(self.testvalue),
                             mayan_long_count_date(12, 16, 11, 16, 9))
            self.assertEqual(mayan_long_count_from_fixed(0),
                             mayan_long_count_date(7, 17, 18, 13, 2))
            self.assertEqual(mayan_haab_from_fixed(self.testvalue),
                             mayan_haab_date(11, 7))
            self.assertEqual(mayan_tzolkin_from_fixed(self.testvalue),
                             mayan_tzolkin_date(11, 9))

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue,
                fixed_from_mayan_long_count(
                    mayan_long_count_date(12, 16, 11, 16, 9)))
            self.assertEqual(
                rd(0),
                fixed_from_mayan_long_count(
                    mayan_long_count_date(7, 17, 18, 13, 2)))
            self.assertEqual(
                mayan_haab_on_or_before(mayan_haab_date(11, 7), self.testvalue),
                self.testvalue)
            self.assertEqual(
                mayan_tzolkin_on_or_before(
                    mayan_tzolkin_date(11, 9), self.testvalue),
                self.testvalue)

    class AztecSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                aztec_xihuitl_from_fixed(self.testvalue),
                aztec_xihuitl_date(2, 1))

        def testConversionToFixed(self):
            self.assertEqual(
                aztec_xihuitl_on_or_before(
                    aztec_xihuitl_date(2, 1), self.testvalue),
                self.testvalue)

```

This definition is continued in chunk 162.

This code is used in chunks 4 and 85.

Defines:

AztecSmokeTestCase, never used.

MayanSmokeTestCase, never used.

Uses aztec_xihuitl_date 83, aztec_xihuitl_from_fixed 83, aztec_xihuitl_on_or_before 83, fixed_from_mayan_long_count 83, mayan_haab_date 83, mayan_haab_from_fixed 83, mayan_haab_on_or_before 83, mayan_long_count_date 83, mayan_long_count_from_fixed 83, mayan_tzolkin_date 83, mayan_tzolkin_from_fixed 83, mayan_tzolkin_on_or_before 83, and rd 37.

2.11.1 Unit tests

```
85  <mayanCalendarsUnitTest.py 85>≡  
    # <generated code warning 1>  
    <import for testing 6>  
    from appendixCUnitTest import AppendixCTable2TestCaseBase  
    <mayan calendars unit test 84>  
    <execute tests 5>
```

Root chunk (not used in this document).
Uses AppendixCTable2TestCaseBase 155.

2.12 Old Hindu Calendars

```
86  <old hindu calendars 86>≡
#####
# old hindu calendars algorithms #
#####
# see lines 2321-2325 in calendrica-3.0.cl
def old_hindu_lunar_date(year, month, leap, day):
    """Return an Old Hindu lunar date data structure."""
    return [year, month, leap, day]

# see lines 2327-2329 in calendrica-3.0.cl
def old_hindu_lunar_month(date):
    """Return the month field of an Old Hindu lunar
    date = [year, month, leap, day]."""
    return date[1]

# see lines 2331-2333 in calendrica-3.0.cl
def old_hindu_lunar_leap(date):
    """Return the leap field of an Old Hindu lunar
    date = [year, month, leap, day]."""
    return date[2]

# see lines 2335-2337 in calendrica-3.0.cl
def old_hindu_lunar_day(date):
    """Return the day field of an Old Hindu lunar
    date = [year, month, leap, day]."""
    return date[3]

# see lines 2339-2341 in calendrica-3.0.cl
def old_hindu_lunar_year(date):
    """Return the year field of an Old Hindu lunar
    date = [year, month, leap, day]."""
    return date[0]

# see lines 2343-2346 in calendrica-3.0.cl
def hindu_solar_date(year, month, day):
    """Return an Hindu solar date data structure."""
    return [year, month, day]

# see lines 2348-2351 in calendrica-3.0.cl
HINDU_EPOCH = fixed_from_julian(julian_date(bce(3102), FEBRUARY, 18))

# see lines 2353-2356 in calendrica-3.0.cl
def hindu_day_count(date):
    """Return elapsed days (Ahargana) to date date since Hindu epoch (KY)."""
    return date - HINDU_EPOCH

# see lines 2358-2361 in calendrica-3.0.cl
ARYA_SOLAR_YEAR = 1577917500/4320000

# see lines 2363-2366 in calendrica-3.0.cl
ARYA_SOLAR_MONTH = ARYA_SOLAR_YEAR / 12

# see lines 2368-2378 in calendrica-3.0.cl
def old_hindu_solar_from_fixed(date):
    """Return Old Hindu solar date equivalent to fixed date date."""
    sun = hindu_day_count(date) + hr(6)
    year = quotient(sun, ARYA_SOLAR_YEAR)
    month = mod(quotient(sun, ARYA_SOLAR_MONTH), 12) + 1
```



```

    day = ifloor(mod(sun, ARYA_SOLAR_MONTH)) + 1
    return hindu_solar_date(year, month, day)

# see lines 2380-2390 in calendrica-3.0.cl
# The following
#     from math import ceil as ceiling
# is not ok, the corresponding CL code
# uses CL ceiling which always returns an integer, while
# ceil from math module always returns a float...so I redefine it
def ceiling(n):
    """Return the integer rounded towards +infinity of n."""
    from math import ceil
    return int(ceil(n))

def fixed_from_old_hindu_solar(s_date):
    """Return fixed date corresponding to Old Hindu solar date s_date."""
    month = standard_month(s_date)
    day = standard_day(s_date)
    year = standard_year(s_date)
    return ceiling(HINDU_EPOCH +
                  year * ARYA_SOLAR_YEAR +
                  (month - 1) * ARYA_SOLAR_MONTH +
                  day + hr(-30))

# see lines 2392-2395 in calendrica-3.0.cl
ARYA_LUNAR_MONTH = 1577917500/53433336

# see lines 2397-2400 in calendrica-3.0.cl
ARYA_LUNAR_DAY = ARYA_LUNAR_MONTH / 30

# see lines 2402-2409 in calendrica-3.0.cl
def is_old_hindu_lunar_leap_year(l_year):
    """Return True if l_year is a leap year on the
    old Hindu calendar."""
    return mod(l_year * ARYA_SOLAR_YEAR - ARYA_SOLAR_MONTH,
              ARYA_LUNAR_MONTH) >= 23902504679/1282400064

# see lines 2411-2431 in calendrica-3.0.cl
def old_hindu_lunar_from_fixed(date):
    """Return Old Hindu lunar date equivalent to fixed date date."""
    sun = hindu_day_count(date) + hr(6)
    new_moon = sun - mod(sun, ARYA_LUNAR_MONTH)
    leap = (((ARYA_SOLAR_MONTH - ARYA_LUNAR_MONTH)
            >=
            mod(new_moon, ARYA_SOLAR_MONTH))
            and
            (mod(new_moon, ARYA_SOLAR_MONTH) > 0))
    month = mod(ceiling(new_moon / ARYA_SOLAR_MONTH), 12) + 1
    day = mod(quotient(sun, ARYA_LUNAR_DAY), 30) + 1
    year = ceiling((new_moon + ARYA_SOLAR_MONTH) / ARYA_SOLAR_YEAR) - 1
    return old_hindu_lunar_date(year, month, leap, day)

# see lines 2433-2460 in calendrica-3.0.cl
def fixed_from_old_hindu_lunar(l_date):
    """Return fixed date corresponding to Old Hindu lunar date l_date."""
    year = old_hindu_lunar_year(l_date)
    month = old_hindu_lunar_month(l_date)
    leap = old_hindu_lunar_leap(l_date)
    day = old_hindu_lunar_day(l_date)
    mina = ((12 * year) - 1) * ARYA_SOLAR_MONTH
    lunar_new_year = ARYA_LUNAR_MONTH * (quotient(mina, ARYA_LUNAR_MONTH) + 1)

```

```

    if ((not leap) and
        (ceiling((lunar_new_year - mina) / (ARYA_SOLAR_MONTH - ARYA_LUNAR_MONTH))
         <= month)):
        temp = month
    else:
        temp = month - 1
    temp = (HINDU_EPOCH +
            lunar_new_year +
            (ARYA_LUNAR_MONTH * temp) +
            ((day - 1) * ARYA_LUNAR_DAY) +
            hr(-6))
    return ceiling(temp)

# see lines 2462-2466 in calendrica-3.0.cl
ARYA_JOVIAN_PERIOD = 1577917500/364224

# see lines 2468-2473 in calendrica-3.0.cl
def jovian_year(date):
    """Return year of Jupiter cycle at fixed date date."""
    return amod(quotient(hindu_day_count(date), ARYA_JOVIAN_PERIOD / 12) + 27,
                60)

```

This code is used in chunk 3.

Defines:

ARYA_JOVIAN_PERIOD, never used.
 ARYA_LUNAR_DAY, never used.
 ARYA_LUNAR_MONTH, never used.
 ARYA_SOLAR_MONTH, never used.
 ARYA_SOLAR_YEAR, never used.
 ceiling, used in chunks 112, 123, 126, and 129.
 fixed_from_old_hindu_lunar, used in chunks 87 and 179.
 fixed_from_old_hindu_solar, used in chunks 87 and 179.
 hindu_day_count, never used.
 HINDU_EPOCH, used in chunk 126.
 hindu_solar_date, used in chunks 87, 126, and 175.
 is_old_hindu_lunar_leap_year, never used.
 jovian_year, used in chunk 87.
 old_hindu_lunar_date, used in chunks 87 and 175.
 old_hindu_lunar_day, never used.
 old_hindu_lunar_from_fixed, used in chunks 87 and 179.
 old_hindu_lunar_leap, never used.
 old_hindu_lunar_month, never used.
 old_hindu_lunar_year, never used.
 old_hindu_solar_from_fixed, used in chunks 87 and 179.

Uses amod 15, bce 59, epoch 37, FEBRUARY 53, fixed_from_julian 59, hr 92, ifloor 15, julian_date 59,
 mod 15, quotient 14 14, standard_day 39, standard_month 39, and standard_year 39.

```

87  <old hindu calendars unit test 87>≡
    class OldHinduSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                old_hindu_solar_from_fixed(self.testvalue),
                hindu_solar_date(5046, 7, 29))
            self.assertEqual(
                old_hindu_lunar_from_fixed(self.testvalue),
                old_hindu_lunar_date(5046, 8, False, 8))
            # FIXME (not sure the check is correct)
            self.assertEqual(jovian_year(self.testvalue), 32)

        def testConversionToFixed(self):
            self.assertEqual(
                self.testvalue,
                fixed_from_old_hindu_solar(hindu_solar_date(5046, 7, 29)))
            self.assertEqual(
                self.testvalue,
                fixed_from_old_hindu_lunar(
                    old_hindu_lunar_date(5046, 8, False, 8)))

```

This definition is continued in chunk 178.

This code is used in chunks 4 and 88.

Defines:

OldHinduSmokeTestCase, never used.

Uses fixed_from_old_hindu_lunar 86, fixed_from_old_hindu_solar 86, hindu_solar_date 86, jovian_year 86, old_hindu_lunar_date 86, old_hindu_lunar_from_fixed 86, and old_hindu_solar_from_fixed 86.

2.12.1 Unit tests

```

88  <oldHinduCalendarsUnitTest.py 88>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable4TestCaseBase
    <old hindu calendars unit test 87>
    <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable4TestCaseBase 175.

2.13 Balinese Calendar

```
89  <balinese calendar 89>≡
#####
# balinese calendar algorithms #
#####
# see lines 2478-2481 in calendrica-3.0.cl
def balinese_date(b1, b2, b3, b4, b5, b6, b7, b8, b9, b0):
    """Return a Balinese date data structure."""
    return [b1, b2, b3, b4, b5, b6, b7, b8, b9, b0]

# see lines 2483-2485 in calendrica-3.0.cl
def bali_luang(b_date):
    return b_date[0]

# see lines 2487-2489 in calendrica-3.0.cl
def bali_dwiwara(b_date):
    return b_date[1]

# see lines 2491-2493 in calendrica-3.0.cl
def bali_triwara(b_date):
    return b_date[2]

# see lines 2495-2497 in calendrica-3.0.cl
def bali_caturwara(b_date):
    return b_date[3]

# see lines 2499-2501 in calendrica-3.0.cl
def bali_pancawara(b_date):
    return b_date[4]

# see lines 2503-2505 in calendrica-3.0.cl
def bali_sadwara(b_date):
    return b_date[5]

# see lines 2507-2509 in calendrica-3.0.cl
def bali_saptawara(b_date):
    return b_date[6]

# see lines 2511-2513 in calendrica-3.0.cl
def bali_asatawara(b_date):
    return b_date[7]

# see lines 2513-2517 in calendrica-3.0.cl
def bali_sangawara(b_date):
    return b_date[8]

# see lines 2519-2521 in calendrica-3.0.cl
def bali_dasawara(b_date):
    return b_date[9]

# see lines 2523-2526 in calendrica-3.0.cl
BALI_EPOCH = fixed_from_jd(146)

# see lines 2528-2531 in calendrica-3.0.cl
def bali_day_from_fixed(date):
    """Return the position of date date in 210_day Pawukon cycle."""
    return mod(date - BALI_EPOCH, 210)

def even(i):
    return mod(i, 2) == 0
```

```

def odd(i):
    return not even(i)

# see lines 2533-2536 in calendrica-3.0.cl
def bali_luang_from_fixed(date):
    """Check membership of date date in "1_day" Balinese cycle."""
    return even(bali_dasawara_from_fixed(date))

# see lines 2538-2541 in calendrica-3.0.cl
def bali_dwiwara_from_fixed(date):
    """Return the position of date date in 2_day Balinese cycle."""
    return amod(bali_dasawara_from_fixed(date), 2)

# see lines 2543-2546 in calendrica-3.0.cl
def bali_triwara_from_fixed(date):
    """Return the position of date date in 3_day Balinese cycle."""
    return mod(bali_day_from_fixed(date), 3) + 1

# see lines 2548-2551 in calendrica-3.0.cl
def bali_caturwara_from_fixed(date):
    """Return the position of date date in 4_day Balinese cycle."""
    return amod(bali_asatawara_from_fixed(date), 4)

# see lines 2553-2556 in calendrica-3.0.cl
def bali_pancawara_from_fixed(date):
    """Return the position of date date in 5_day Balinese cycle."""
    return amod(bali_day_from_fixed(date) + 2, 5)

# see lines 2558-2561 in calendrica-3.0.cl
def bali_sadwara_from_fixed(date):
    """Return the position of date date in 6_day Balinese cycle."""
    return mod(bali_day_from_fixed(date), 6) + 1

# see lines 2563-2566 in calendrica-3.0.cl
def bali_saptawara_from_fixed(date):
    """Return the position of date date in Balinese week."""
    return mod(bali_day_from_fixed(date), 7) + 1

# see lines 2568-2576 in calendrica-3.0.cl
def bali_asatawara_from_fixed(date):
    """Return the position of date date in 8_day Balinese cycle."""
    day = bali_day_from_fixed(date)
    return mod(max(6, 4 + mod(day - 70, 210)), 8) + 1

# see lines 2578-2583 in calendrica-3.0.cl
def bali_sangawara_from_fixed(date):
    """Return the position of date date in 9_day Balinese cycle."""
    return mod(max(0, bali_day_from_fixed(date) - 3), 9) + 1

# see lines 2585-2594 in calendrica-3.0.cl
def bali_dasawara_from_fixed(date):
    """Return the position of date date in 10_day Balinese cycle."""
    i = bali_pancawara_from_fixed(date) - 1
    j = bali_saptawara_from_fixed(date) - 1
    return mod(1 + [5, 9, 7, 4, 8][i] + [5, 4, 3, 7, 8, 6, 9][j], 10)

# see lines 2596-2609 in calendrica-3.0.cl
def bali_pawukon_from_fixed(date):
    """Return the positions of date date in ten cycles of Balinese Pawukon
    calendar."""

```

```

    return balinese_date(bali_luang_from_fixed(date),
                          bali_dwiwara_from_fixed(date),
                          bali_triwara_from_fixed(date),
                          bali_caturwara_from_fixed(date),
                          bali_pancawara_from_fixed(date),
                          bali_sadwara_from_fixed(date),
                          bali_saptawara_from_fixed(date),
                          bali_asatawara_from_fixed(date),
                          bali_sangawara_from_fixed(date),
                          bali_dasawara_from_fixed(date))

# see lines 2611-2614 in calendrica-3.0.cl
def bali_week_from_fixed(date):
    """Return the week number of date date in Balinese cycle."""
    return quotient(bali_day_from_fixed(date), 7) + 1

# see lines 2616-2630 in calendrica-3.0.cl
def bali_on_or_before(b_date, date):
    """Return last fixed date on or before date with Pawukon date b_date."""
    a5 = bali_pancawara(b_date) - 1
    a6 = bali_sadwara(b_date) - 1
    b7 = bali_saptawara(b_date) - 1
    b35 = mod(a5 + 14 + (15 * (b7 - a5)), 35)
    days = a6 + (36 * (b35 - a6))
    cap_Delta = bali_day_from_fixed(0)
    return date - mod(date + cap_Delta - days, 210)

# see lines 2632-2646 in calendrica-3.0.cl
def positions_in_range(n, c, cap_Delta, range):
    """Return the list of occurrences of n-th day of c-day cycle
    in range.
    cap_Delta is the position in cycle of RD 0."""
    a = start(range)
    b = end(range)
    pos = a + mod(n - a - cap_Delta - 1, c)
    return (nil if (pos > b) else
            [pos].extend(
                positions_in_range(n, c, cap_Delta, interval(pos + 1, b))))

# see lines 2648-2654 in calendrica-3.0.cl
def kajeng_keliwon(g_year):
    """Return the occurrences of Kajeng Keliwon (9th day of each
    15_day subcycle of Pawukon) in Gregorian year g_year."""
    year = gregorian_year_range(g_year)
    cap_Delta = bali_day_from_fixed(0)
    return positions_in_range(9, 15, cap_Delta, year)

# see lines 2656-2662 in calendrica-3.0.cl
def tumpek(g_year):
    """Return the occurrences of Tumpek (14th day of Pawukon and every
    35th subsequent day) within Gregorian year g_year."""
    year = gregorian_year_range(g_year)
    cap_Delta = bali_day_from_fixed(0)
    return positions_in_range(14, 35, cap_Delta, year)

```

This code is used in chunk 3.

Defines:

```

bali_asatawara, never used.
bali_asatawara_from_fixed, never used.
bali_caturwara, never used.
bali_caturwara_from_fixed, never used.
bali_dasawara, never used.

```

bali_dasawara_from_fixed, never used.
 bali_day_from_fixed, never used.
 bali_dwiwara, never used.
 bali_dwiwara_from_fixed, never used.
 BALI_EPOCH, never used.
 bali_luang, never used.
 bali_luang_from_fixed, never used.
 bali_on_or_before, used in chunk 90.
 bali_pancawara, never used.
 bali_pancawara_from_fixed, never used.
 bali_pawukon_from_fixed, used in chunks 90 and 170.
 bali_sadwara, never used.
 bali_sadwara_from_fixed, never used.
 bali_sangawara, never used.
 bali_sangawara_from_fixed, never used.
 bali_saptawara, never used.
 bali_saptawara_from_fixed, never used.
 bali_triwara, never used.
 bali_triwara_from_fixed, never used.
 bali_week_from_fixed, never used.
 balinese_date, used in chunks 90 and 164.
 even, never used.
 kajeng_keliwon, never used.
 odd, used in chunk 123.
 positions_in_range, never used.
 tumpek, never used.
 Uses amod 15, end 47, fixed_from_jd 48, gregorian_year_range 56, interval 47, mod 15, quotient 14 14, and start 47.

```

90  <balinese calendar unit test 90>≡
    class BalineseSmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.testvalue = 710347

        def testConversionFromFixed(self):
            self.assertEqual(
                bali_pawukon_from_fixed(self.testvalue),
                balinese_date(True, 2, 1, 1, 3, 1, 2, 5, 7, 2))

        def testConversionToFixed(self):
            self.assertEqual(
                bali_on_or_before(
                    balinese_date(True, 2, 1, 1, 3, 1, 2, 5, 7, 2),
                    self.testvalue),
                self.testvalue)
  
```

This definition is continued in chunk 169.

This code is used in chunks 4 and 91.

Defines:

BalineseSmokeTestCase, never used.

Uses bali_on_or_before 89, bali_pawukon_from_fixed 89, and balinese_date 89.

2.13.1 Unit tests

```

91  <balineseCalendarUnitTest.py 91>≡
    # <generated code warning 1>
    <import for testing 6>
    from appendixCUnitTest import AppendixCTable3TestCaseBase
    <balinese calendar unit test 90>
    <execute tests 5>
  
```

Root chunk (not used in this document).

Uses AppendixCTable3TestCaseBase 164.

2.14 Time and Astronomy

The location of an object in the sky is determined by celestial coordinates, analogous to the latitude and longitude for the location of a position on Earth.

2.14.1 *alt – az* Coordinate systems

The alt-az is a topocentric (i.e. as seen from the observer's place on the Earth's surface) celestial coordinate system. It uses the horizon as its fundamental circle which divides the sky in two hemispheres. The pole of the upper hemisphere is called the zenith, while the one of the lower hemisphere is called nadir. These are defined by the local vertical (using a plumb-line or similar). The point of origin on the horizon is determined by the intersection of the vertical circle (i.e. through zenith and nadir) passing through north and south celestial pole.

The azimuth (A) is the direction of a celestial object, measured clockwise around the observer's horizon from south.

2.14.2 *HA – dec* Coordinate systems

2.14.3 *RA – dec* Coordinate systems

Figure 2.1 displays the celestial sphere and the RA-dec coordinate system (also named second equatorial).

Figure 2.2 displays ecliptic, lunar orbit (angle is exaggerated), the lunar (ascending) node Ω and the origin Υ (at vernal equinox). It is interesting to note graphically what is later described algorithmically.

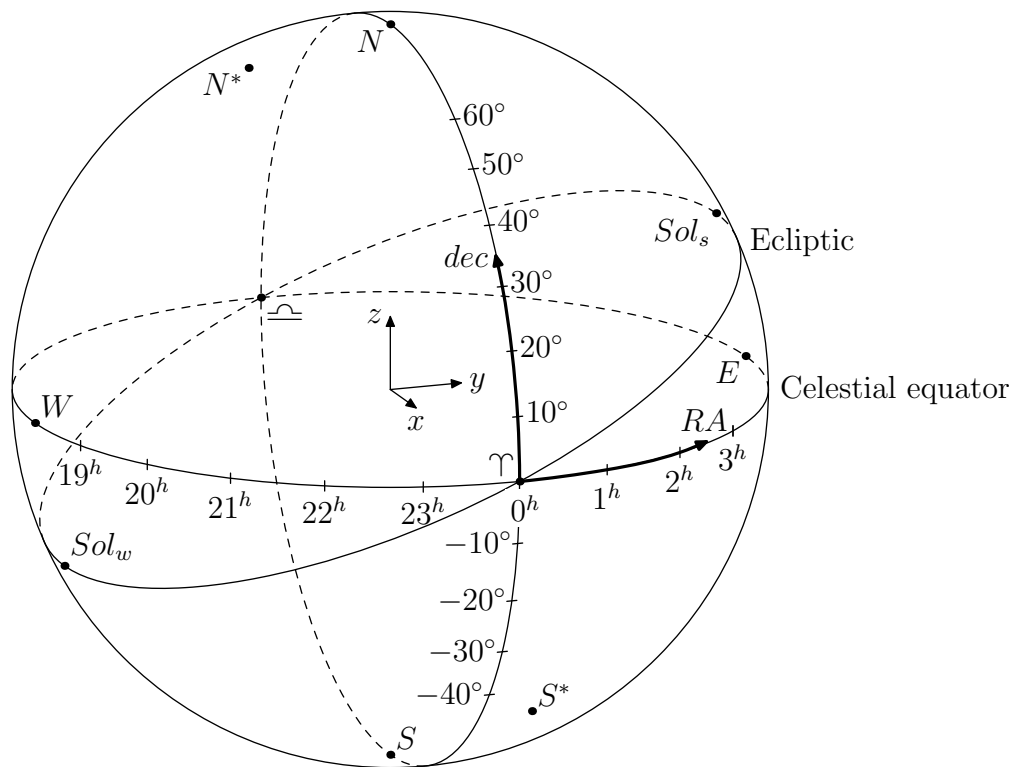


Figure 2.1: RA-dec coordinate system. N^* is the North pole of the ecliptic; N is the North Celestial Pole; Υ is the first point of Aries and the zero-point for this coordinate system. Right Ascension (RA or α) is measured in hours towards East (E); declination (dec or δ) is measured in degrees, positive if north (N) of the Celestial equator. Sol_w is the winter solstice point of $18^h RA$ and $-23.5^\circ dec$; Sol_s is the summer solstice point of $6^h RA$ and $23.5^\circ dec$.

```

92  <time and astronomy 92>≡
    #####
    # Time and Astronomy #
    #####
    # see lines 2667-2670 in calendrica-3.0.cl
    def hr(x):
        """Return the number of days given x hours."""
        return x / 24

    # see lines 2672-2675 in calendrica-3.0.cl
    def sec(x):
        """Return the number of days given x seconds."""
        return x / 24 / 60 / 60

    # see lines 2677-2680 in calendrica-3.0.cl
    def mt(x):
        """Return x as meters."""
        return x

    # see lines 2682-2686 in calendrica-3.0.cl
    def deg(x):
        """Return the degrees in angle x."""
        return x

```

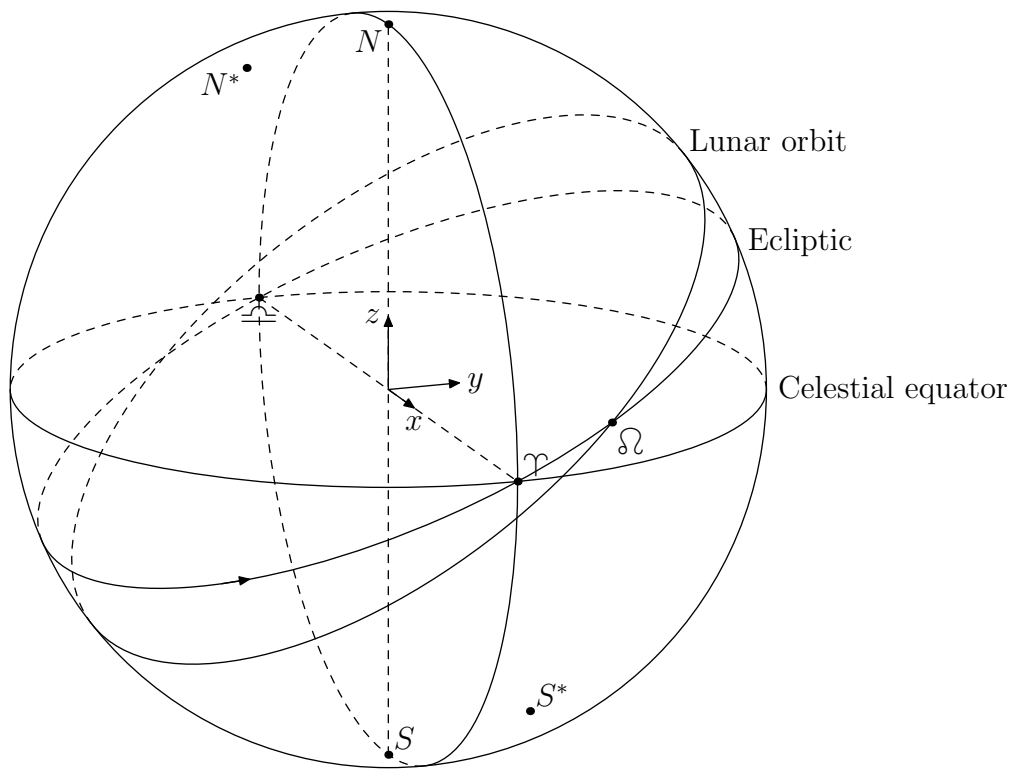


Figure 2.2: Celestial coordinate system. N^* is the North pole of the ecliptic; N is the North Celestial Pole; Υ is the first point of Aries or the ascending node of the mean ecliptic. Ω is the ascending node of the lunar orbit.

```
# see lines 2688-2690 in calendrica-3.0.cl
def secs(x):
    """Return the seconds in angle x."""
    return x / 3600

# see lines 2692-2696 in calendrica-3.0.cl
def angle(d, m, s):
    """Return an angle data structure
    from d degrees, m arcminutes and s arcseconds.
    This assumes that negative angles specifies negative d, m and s."""
    return d + ((m + (s / 60)) / 60)

# see lines 2698-2701 in calendrica-3.0.cl
def degrees(theta):
    """Return a normalize angle theta to range [0,360) degrees."""
    return mod(theta, 360)

# see lines 2703-2706 in calendrica-3.0.cl
def degrees_from_radians(theta):
    from math import degrees as math_degrees
    return degrees(math_degrees(theta))

# see lines 2708-2711 in calendrica-3.0.cl
def radians_from_degrees(theta):
    pass
```

```

from math import radians as radians_from_degrees

# see lines 2713-2716 in calendrica-3.0.cl
def sin_degrees(theta):
    """Return sine of theta (given in degrees)."""
    from math import sin
    return sin(radians_from_degrees(theta))

# see lines 2718-2721 in calendrica-3.0.cl
def cosine_degrees(theta):
    """Return cosine of theta (given in degrees)."""
    from math import cos
    return cos(radians_from_degrees(theta))

# see lines 2723-2726 in calendrica-3.0.cl
def tangent_degrees(theta):
    """Return tangent of theta (given in degrees)."""
    return tan(radians_from_degrees(theta))

def signum(a):
    if a > 0:
        return 1
    elif a == 0:
        return 0
    else:
        return -1

#-----
# NOTE: arc[tan|sin|cos] casted with degrees given CL code
#       returns angles [0, 360), see email from Dershowitz
#       after my request for clarification
#-----

# see lines 2728-2739 in calendrica-3.0.cl
# def arctan_degrees(y, x):
#     """ Arctangent of y/x in degrees."""
#     from math import atan2
#     return degrees(degrees_from_radians(atan2(x, y)))

def arctan_degrees(y, x):
    """ Arctangent of y/x in degrees."""
    if (x == 0) and (y != 0):
        return mod(signum(y) * deg(mpf(90)), 360)
    else:
        alpha = degrees_from_radians(atan(y / x))
        if x >= 0:
            return alpha
        else:
            return mod(alpha + deg(mpf(180)), 360)

# see lines 2741-2744 in calendrica-3.0.cl
def arcsin_degrees(x):
    """Return arcsine of x in degrees."""
    from math import asin
    return degrees(degrees_from_radians(asin(x)))

# see lines 2746-2749 in calendrica-3.0.cl
def arccos_degrees(x):
    """Return arccosine of x in degrees."""

```

```

    from math import acos
    return degrees(degrees_from_radians(acos(x)))

# see lines 2751-2753 in calendrica-3.0.cl
def location(latitude, longitude, elevation, zone):
    """Return a location data structure."""
    return [latitude, longitude, elevation, zone]

# see lines 2755-2757 in calendrica-3.0.cl
def latitude(location):
    """Return the latitude field of a location."""
    return location[0]

# see lines 2759-2761 in calendrica-3.0.cl
def longitude(location):
    """Return the longitude field of a location."""
    return location[1]

# see lines 2763-2765 in calendrica-3.0.cl
def elevation(location):
    """Return the elevation field of a location."""
    return location[2]

# see lines 2767-2769 in calendrica-3.0.cl
def zone(location):
    """Return the timezone field of a location."""
    return location[3]

# see lines 2771-2775 in calendrica-3.0.cl
MECCA = location(angle(21, 25, 24), angle(39, 49, 24), mt(298), hr(3))

# see lines 5898-5901 in calendrica-3.0.cl
JERUSALEM = location(31.8, 35.2, mt(800), hr(2))

BRUXELLES = location(angle(4, 21, 17), angle(50, 50, 47), mt(800), hr(1))

URBANA = location(40.1,
                  -88.2,
                  mt(225),
                  hr(-6))

GREENWHICH = location(51.4777815,
                      0,
                      mt(46.9),
                      hr(0))

# see lines 2777-2797 in calendrica-3.0.cl
def direction(location, focus):
    """Return the angle (clockwise from North) to face focus when
    standing in location, location. Subject to errors near focus and
    its antipode."""
    phi = latitude(location)
    phi_prime = latitude(focus)
    psi = longitude(location)
    psi_prime = longitude(focus)
    y = sin_degrees(psi_prime - psi)
    x = ((cosine_degrees(phi) * tangent_degrees(phi_prime)) -
         (sin_degrees(phi) * cosine_degrees(psi - psi_prime)))
    if ((x == y == 0) or (phi_prime == deg(90))):
        return deg(0)

```

```

elif (phi_prime == deg(-90)):
    return deg(180)
else:
    return arctan_degrees(y, x)

# see lines 2799-2803 in calendrica-3.0.cl
def standard_from_universal(tee_rom_u, location):
    """Return standard time from tee_rom_u in universal time at location."""
    return tee_rom_u + zone(location)

# see lines 2805-2809 in calendrica-3.0.cl
def universal_from_standard(tee_rom_s, location):
    """Return universal time from tee_rom_s in standard time at location."""
    return tee_rom_s - zone(location)

# see lines 2811-2815 in calendrica-3.0.cl
def zone_from_longitude(phi):
    """Return the difference between UT and local mean time at longitude
    'phi' as a fraction of a day."""
    return phi / deg(360)

# see lines 2817-2820 in calendrica-3.0.cl
def local_from_universal(tee_rom_u, location):
    """Return local time from universal tee_rom_u at location, location."""
    return tee_rom_u + zone_from_longitude(longitude(location))

# see lines 2822-2825 in calendrica-3.0.cl
def universal_from_local(tee_ell, location):
    """Return universal time from local tee_ell at location, location."""
    return tee_ell - zone_from_longitude(longitude(location))

# see lines 2827-2832 in calendrica-3.0.cl
def standard_from_local(tee_ell, location):
    """Return standard time from local tee_ell at loacle, location."""
    return standard_from_universal(universal_from_local(tee_ell, location),
                                   location)

# see lines 2834-2839 in calendrica-3.0.cl
def local_from_standard(tee_rom_s, location):
    """Return local time from standard tee_rom_s at location, location."""
    return local_from_universal(universal_from_standard(tee_rom_s, location),
                                location)

# see lines 2841-2844 in calendrica-3.0.cl
def apparent_from_local(tee, location):
    """Return sundial time at local time tee at location, location."""
    return tee + equation_of_time(universal_from_local(tee, location))

# see lines 2846-2849 in calendrica-3.0.cl
def local_from_apparent(tee, location):
    """Return local time from sundial time tee at location, location."""
    return tee - equation_of_time(universal_from_local(tee, location))

# see lines 2851-2857 in calendrica-3.0.cl
def midnight(date, location):
    """Return standard time on fixed date, date, of true (apparent)
    midnight at location, location."""
    return standard_from_local(local_from_apparent(date, location), location)

# see lines 2859-2864 in calendrica-3.0.cl
def midday(date, location):

```

```

        """Return standard time on fixed date, date, of midday
        at location, location."""
        return standard_from_local(local_from_apparent(date + hr(mpf(12)),
                                                         location), location)

# see lines 2866-2870 in calendrica-3.0.c1
def julian_centuries(tee):
    """Return Julian centuries since 2000 at moment tee."""
    return (dynamical_from_universal(tee) - J2000) / mpf(36525)

# see lines 2872-2880 in calendrica-3.0.c1
def obliquity(tee):
    """Return obliquity of ecliptic at moment tee."""
    c = julian_centuries(tee)
    return (angle(23, 26, mpf(21.448)) +
            poly(c, [mpf(0),
                    angle(0, 0, mpf(-46.8150)),
                    angle(0, 0, mpf(-0.00059)),
                    angle(0, 0, mpf(0.001813))]))

# see lines 2882-2891 in calendrica-3.0.c1
def declination(tee, beta, lam):
    """Return declination at moment UT tee of object at
    longitude 'lam' and latitude 'beta'."""
    varepsilon = obliquity(tee)
    return arcsin_degrees(
        (sin_degrees(beta) * cosine_degrees(varepsilon)) +
        (cosine_degrees(beta) * sin_degrees(varepsilon) * sin_degrees(lam)))

# see lines 2893-2903 in calendrica-3.0.c1
def right_ascension(tee, beta, lam):
    """Return right ascension at moment UT 'tee' of object at
    latitude 'lam' and longitude 'beta'."""
    varepsilon = obliquity(tee)
    return arctan_degrees(
        (sin_degrees(lam) * cosine_degrees(varepsilon)) -
        (tangent_degrees(beta) * sin_degrees(varepsilon)),
        cosine_degrees(lam))

# see lines 2905-2920 in calendrica-3.0.c1
def sine_offset(tee, location, alpha):
    """Return sine of angle between position of sun at
    local time tee and when its depression is alpha at location, location.
    Out of range when it does not occur."""
    phi = latitude(location)
    tee_prime = universal_from_local(tee, location)
    delta = declination(tee_prime, deg(mpf(0)), solar_longitude(tee_prime))
    return ((tangent_degrees(phi) * tangent_degrees(delta)) +
            (sin_degrees(alpha) / (cosine_degrees(delta) *
                                   cosine_degrees(phi))))

# see lines 2922-2947 in calendrica-3.0.c1
def approx_moment_of_depression(tee, location, alpha, early):
    """Return the moment in local time near tee when depression angle
    of sun is alpha (negative if above horizon) at location;
    early is true when MORNING event is sought and false for EVENING.
    Returns BOGUS if depression angle is not reached."""
    ttry = sine_offset(tee, location, alpha)
    date = fixed_from_moment(tee)

    if (alpha >= 0):

```

```

        if early:
            alt = date
        else:
            alt = date + 1
    else:
        alt = date + hr(12)

    if (abs(ttry) > 1):
        value = sine_offset(alt, location, alpha)
    else:
        value = ttry

    if (abs(value) <= 1):
        temp = -1 if early else 1
        temp *= mod(hr(12) + arcsin_degrees(value) / deg(360), 1) - hr(6)
        temp += date + hr(12)
        return local_from_apparent(temp, location)
    else:
        return BOGUS

# see lines 2949-2963 in calendrica-3.0.cl
def moment_of_depression(approx, location, alpha, early):
    """Return the moment in local time near approx when depression
    angle of sun is alpha (negative if above horizon) at location;
    early is true when MORNING event is sought, and false for EVENING.
    Returns BOGUS if depression angle is not reached."""
    tee = approx_moment_of_depression(approx, location, alpha, early)
    if (tee == BOGUS):
        return BOGUS
    else:
        if (abs(approx - tee) < sec(30)):
            return tee
        else:
            return moment_of_depression(tee, location, alpha, early)

# see lines 2965-2968 in calendrica-3.0.cl
MORNING = True

# see lines 2970-2973 in calendrica-3.0.cl
EVENING = False

# see lines 2975-2984 in calendrica-3.0.cl
def dawn(date, location, alpha):
    """Return standard time in morning on fixed date date at
    location location when depression angle of sun is alpha.
    Returns BOGUS if there is no dawn on date date."""
    result = moment_of_depression(date + hr(6), location, alpha, MORNING)
    if (result == BOGUS):
        return BOGUS
    else:
        return standard_from_local(result, location)

# see lines 2986-2995 in calendrica-3.0.cl
def dusk(date, location, alpha):
    """Return standard time in evening on fixed date 'date' at
    location 'location' when depression angle of sun is alpha.
    Return BOGUS if there is no dusk on date 'date'."""
    result = moment_of_depression(date + hr(18), location, alpha, EVENING)
    if (result == BOGUS):
        return BOGUS

```

```

else:
    return standard_from_local(result, location)

from math import sqrt
h = max(mt(0), elevation(location))
cap_R = mt(6.372E6)
dip = arccos_degrees(cap_R / (cap_R + h))
alpha = angle(0, 50, 0) + dip + secs(19) * sqrt(h)

# see lines 440-451 in calendrica-3.0.errata.cl
def refraction(tee, location):
    """Return refraction angle at location 'location' and time 'tee'."""
    from math import sqrt
    h = max(mt(0), elevation(location))
    cap_R = mt(6.372E6)
    dip = arccos_degrees(cap_R / (cap_R + h))
    return angle(0, 50, 0) + dip + secs(19) * sqrt(h)

# see lines 2997-3007 in calendrica-3.0.cl
def sunrise(date, location):
    """Return Standard time of sunrise on fixed date 'date' at
    location 'location'."""
    alpha = refraction(date, location)
    return dawn(date, location, alpha)

# see lines 3009-3019 in calendrica-3.0.cl
def sunset(date, location):
    """Return standard time of sunset on fixed date 'date' at
    location 'location'."""
    alpha = refraction(date, location)
    return dusk(date, location, alpha)

# see lines 453-458 in calendrica-3.0.errata.cl
def observed_lunar_altitude(tee, location):
    """Return the observed altitude of moon at moment, tee, and
    at location, location, taking refraction into account."""
    return topocentric_lunar_altitude(tee, location) + refraction(tee, location)

# see lines 460-467 in calendrica-3.0.errata.cl
def moonrise(date, location):
    """Return the standard time of moonrise on fixed, date,
    and location, location."""
    t = universal_from_standard(date, location)
    waning = (lunar_phase(t) > deg(180))
    alt = observed_lunar_altitude(t, location)
    offset = alt / 360
    if (waning and (offset > 0)):
        approx = t + 1 - offset
    elif waning:
        approx = t - offset
    else:
        approx = t + (1 / 2) + offset
    rise = binary_search(approx - hr(3),
                        approx + hr(3),
                        lambda u, l: ((u - l) < hr(1 / 60)),
                        lambda x: observed_lunar_altitude(x, location) > deg(0))
    return standard_from_universal(rise, location) if (rise < (t + 1)) else BOGUS

def urbana_sunset(gdate):

```



```

        """Return sunset time in Urbana, Ill, on Gregorian date 'gdate'."""
        return time_from_moment(sunset(fixed_from_gregorian(gdate), URBANA))

# from eq 13.38 pag. 191
def urbana_winter(g_year):
    """Return standard time of the winter solstice in Urbana, Illinois, USA."""
    return standard_from_universal(
        solar_longitude_after(
            WINTER,
            fixed_from_gregorian(gregorian_date(g_year, JANUARY, 1))),
        URBANA)

```

This code is used in chunk 3.

Defines:

angle, used in chunks 30, 46, 95, 97, 109, 110, 118, 123, 126, 132, and 186.
 apparent_from_local, used in chunk 132.
 approx_moment_of_depression, never used.
 arccos_degrees, used in chunk 132.
 arcsin_degrees, used in chunk 110.
 arctan_degrees, used in chunks 95 and 99.
 BRUXELLES, never used.
 cosine_degrees, used in chunks 97, 99, 110, and 132.
 dawn, used in chunks 126, 184, and 186.
 declination, used in chunks 94, 95, and 110.
 deg, used in chunks 94, 95, 97, 99, 101, 103, 105, 107, 109, 110, 112, 115, 123, 126, and 132.
 degrees, used in chunks 46, 99, 101, 103, 105, 107, 109, 110, 123, and 126.
 degrees_from_radians, never used.
 direction, never used.
 dusk, used in chunks 94, 95, 126, and 132.
 elevation, never used.
 EVENING, never used.
 GREENWHICH, never used.
 hr, used in chunks 80, 86, 95, 97, 112, 115, 118, 123, 126, and 132.
 JERUSALEM, used in chunk 186.
 julian_centuries, used in chunks 97, 99, 109, and 110.
 latitude, used in chunks 95, 107, 109, 110, and 126.
 local_from_apparent, never used.
 local_from_standard, never used.
 local_from_universal, used in chunk 132.
 location, used in chunks 95, 110, 112, 115, 118, 123, 126, and 132.
 longitude, used in chunks 97, 99, 109, 110, 123, 126, and 184.
 MECCA, never used.
 midday, used in chunks 95 and 112.
 midnight, used in chunks 118, 123, and 126.
 moment_of_depression, never used.
 moonrise, never used.
 MORNING, never used.
 mt, used in chunks 110, 112, 115, 118, 123, 126, 132, 155, and 163.
 obliquity, used in chunk 97.
 radians_from_degrees, never used.
 refraction, used in chunk 110.
 right_ascension, used in chunks 94 and 110.
 sec, never used.
 secs, used in chunk 99.
 signum, used in chunks 97 and 126.
 sin_degrees, used in chunks 97, 99, 109, 110, and 126.
 sine_offset, never used.
 standard_from_local, never used.
 standard_from_universal, used in chunk 123.
 sunrise, used in chunks 95, 126, and 242.
 sunset, used in chunks 95, 115, 126, 132, 184, and 186.
 tangent_degrees, used in chunks 95 and 97.
 universal_from_local, used in chunk 126.
 universal_from_standard, used in chunks 95, 112, 115, 118, 123, and 132.
 URBANA, never used.
 urbana_sunset, never used.
 urbana_winter, used in chunk 93.
 zone, used in chunk 123.

zone_from_longitude, never used.
Uses binary_search 25, BOGUS 13, dynamical_from_universal 95, equation_of_time 97,
fixed_from_gregorian 55, fixed_from_moment 40, gregorian_date 52, J2000 97, JANUARY 53,
lunar_phase 109, mod 15, poly 34, seconds 39, solar_longitude 97, solar_longitude_after 99,
time_from_moment 40, topocentric_lunar_altitude 110, and WINTER 99.

93 *<astronomical algorithms tests 93>*≡
def testUrbanaWinter(self):
 # from the values in the book pag 191
 self.assertAlmostEqual(
 urbana_winter(2000), 730475.31751, 5)

This code is used in chunk 94.
Defines:
testUrbanaWinter, never used.
Uses urbana_winter 92.

```

94  <time and astronomy unit test 94>≡
    class TimeAndAstronomySmokeTestCase(unittest.TestCase):
        def setUp(self):
            self.rd = [-214193, -61387, 25469, 49217, 171307, 210155, 253427,
                      369740, 400085, 434355, 452605, 470160, 473837, 507850,
                      524156, 544676, 567118, 569477, 601716, 613424, 626596,
                      645554, 664224, 671401, 694799, 704424, 708842, 709409,
                      709580, 727274, 728714, 744313, 764652]

            self.declinations = [341.009933681, 344.223866057, 344.349150723,
                                 343.080796014, 6.111045686, 23.282088850,
                                 11.054626067, 20.772095601, 350.530615797,
                                 26.524557874, 24.624220236, 341.329137381,
                                 22.952455871, 28.356788216, 11.708349719,
                                 17.836387256, 1.234462343, 342.613034686,
                                 339.494416096, 10.077195527, 356.273352051,
                                 10.933004147, 333.162727246, 12.857424363,
                                 342.981182734, 8.352097710, 342.717593219,
                                 359.480653210, 339.868605556, 6.747953072,
                                 15.403930316, 5.935073706, 6.502803786]

            self.right_ascensions = [243.344057675, 204.985406451, 210.404938685,
                                     292.982801046, 157.347243474, 109.710580543,
                                     38.206587532, 99.237553669, 334.622772431,
                                     92.594013257, 77.002562902, 275.265641321,
                                     132.240141523, 89.495057657, 21.938682002,
                                     51.336108524, 189.141475514, 323.504045205,
                                     317.763636501, 146.668234288, 183.868193626,
                                     143.441024476, 251.771505962, 154.432825924,
                                     288.759213491, 24.368877399, 291.218608152,
                                     190.563965149, 285.912816020, 152.814362172,
                                     50.014265486, 26.456502208, 177.918419842]

            self.lunar_altitudes = [-11.580406490, -13.996642398, -72.405467670,
                                    -26.949751162, 60.491536818, -32.333449636,
                                    43.325012802, -28.913935286, 20.844069354,
                                    -9.603298107, -13.290409748, 20.650429381,
                                    -9.068998404, -24.960604514, -34.865669400,
                                    -40.121041983, -50.193172697, -39.456259107,
                                    32.614203610, -46.078519304, -51.828340409,
                                    -42.577971851, -15.990046584, 28.658077283,
                                    22.718206310, 61.618573945, -26.504789606,
                                    32.371736207, -38.544325288, 31.594345546,
                                    -28.348377620, 30.478724056, -43.754783219]

            self.dusks = [-214193.22, -61387.297, 25468.746, 49216.734,
                          171306.7, 210154.78, 253426.7, 369739.78,
                          400084.8, 434354.78, 452604.75, 470159.78,
                          473836.78, 507849.8, 524155.75, 544675.7,
                          567117.7, 569476.7, 601715.75, 613423.75,
                          626595.75, 645553.75, 664223.75, 671400.7,
                          694798.75, 704423.75, 708841.7, 709408.75,
                          709579.7, 727273.7, 728713.7, 744312.7,
                          764651.75]

        def testDeclination(self):
            for i in range(len(self.rd)):
                lamb = lunar_longitude(self.rd[i])
                beta = lunar_latitude(self.rd[i])
                alpha = declination(self.rd[i], beta, lamb)

```

```

        self.assertAlmostEqual(alpha, self.declinations[i], 7)

    def testRightAscension(self):
        for i in range(len(self.rd)):
            lamb = lunar_longitude(self.rd[i])
            beta = lunar_latitude(self.rd[i])
            alpha = right_ascension(self.rd[i], beta, lamb)
            self.assertAlmostEqual(alpha, self.right_ascensions[i], 7)

    def testLunarAltitude(self):
        for i in range(len(self.rd)):
            alpha = lunar_altitude(self.rd[i], JAFFA)
            self.assertAlmostEqual(alpha, self.lunar_altitudes[i], 6)

    def testDusk(self):
        for i in range(len(self.rd)):
            du = dusk(self.rd[i] - 1, JAFFA, deg(mpf(4.5)))
            self.assertAlmostEqual(du, self.dusks[i], 0)

class AstronomicalAlgorithmsTestCase(unittest.TestCase):
    (astronomical algorithms tests 93)

```

This definition is continued in chunk 185.

This code is used in chunks 4 and 111.

Defines:

AstronomicalAlgorithmsTestCase, never used.

TimeAndAstronomySmokeTestCase, never used.

Uses declination 92, deg 92, dusk 92, JAFFA 132, lunar_altitude 110, lunar_latitude 109, lunar_longitude 109, rd 37, and right_ascension 92.

```

95  (astronomical lunar calendars 95)≡
#####
# astronomical lunar calendars algorithms #
#####
# see lines 3021-3025 in calendrica-3.0.cl
def jewish_dusk(date, location):
    """Return standard time of Jewish dusk on fixed date, date,
    at location, location, (as per Vilna Gaon)."""
    return dusk(date, location, angle(4, 40, 0))

# see lines 3027-3031 in calendrica-3.0.cl
def jewish_sabbath_ends(date, location):
    """Return standard time of end of Jewish sabbath on fixed date, date,
    at location, location, (as per Berthold Cohn)."""
    return dusk(date, location, angle(7, 5, 0))

# see lines 3033-3042 in calendrica-3.0.cl
def daytime_temporal_hour(date, location):
    """Return the length of daytime temporal hour on fixed date, date
    at location, location.
    Return BOGUS if there no sunrise or sunset on date, date."""
    if (sunrise(date, location) == BOGUS) or (sunset(date, location) == BOGUS):
        return BOGUS
    else:
        return (sunset(date, location) - sunrise(date, location)) / 12

# see lines 3044-3053 in calendrica-3.0.cl
def nighttime_temporal_hour(date, location):
    """Return the length of nighttime temporal hour on fixed date, date,
    at location, location.
    Return BOGUS if there no sunrise or sunset on date, date."""
    if ((sunrise(date + 1, location) == BOGUS) or
        (sunset(date, location) == BOGUS)):
        return BOGUS
    else:
        return (sunrise(date + 1, location) - sunset(date, location)) / 12

# see lines 3055-3073 in calendrica-3.0.cl
def standard_from_sundial(tee, location):
    """Return standard time of temporal moment, tee, at location, location.
    Return BOGUS if temporal hour is undefined that day."""
    date = fixed_from_moment(tee)
    hour = 24 * mod(tee, 1)
    if (6 <= hour <= 18):
        h = daytime_temporal_hour(date, location)
    elif (hour < 6):
        h = nighttime_temporal_hour(date - 1, location)
    else:
        h = nighttime_temporal_hour(date, location)

    # return
    if (h == BOGUS):
        return BOGUS
    elif (6 <= hour <= 18):
        return sunrise(date, location) + ((hour - 6) * h)
    elif (hour < 6):
        return sunset(date - 1, location) + ((hour + 6) * h)
    else:
        return sunset(date, location) + ((hour - 18) * h)

```

```

# see lines 3075-3079 in calendrica-3.0.cl
def jewish_morning_end(date, location):
    """Return standard time on fixed date, date, at location, location,
    of end of morning according to Jewish ritual."""
    return standard_from_sundial(date + hr(10), location)

# see lines 3081-3099 in calendrica-3.0.cl
def asr(date, location):
    """Return standard time of asr on fixed date, date,
    at location, location."""
    noon = universal_from_standard(midday(date, location), location)
    phi = latitude(location)
    delta = declination(noon, deg(0), solar_longitude(noon))
    altitude = delta - phi - deg(90)
    h = arctan_degrees(tangent_degrees(altitude),
        2 * tangent_degrees(altitude) + 1)
    # For Shafii use instead:
    # tangent_degrees(altitude) + 1)

    return dusk(date, location, -h)

##### here start the code inspired by Meeus
# see lines 3101-3104 in calendrica-3.0.cl
def universal_from_dynamical(tee):
    """Return Universal moment from Dynamical time, tee."""
    return tee - ephemeris_correction(tee)

# see lines 3106-3109 in calendrica-3.0.cl
def dynamical_from_universal(tee):
    """Return Dynamical time at Universal moment, tee."""
    return tee + ephemeris_correction(tee)

```

This definition is continued in chunks 97, 99, 101, 103, 105, 107, 109, 110, and 132.

This code is used in chunk 3.

Defines:

- asr, never used.
- daytime_temporal_hour, never used.
- dynamical_from_universal, used in chunks 92 and 96.
- jewish_dusk, never used.
- jewish_morning_end, never used.
- jewish_sabbath_ends, never used.
- nighttime_temporal_hour, never used.
- standard_from_sundial, used in chunk 126.
- universal_from_dynamical, used in chunks 96, 98, and 109.

Uses angle 92, arctan_degrees 92 92, BOGUS 13, declination 92, deg 92, dusk 92, end 47, ephemeris_correction 97, fixed_from_moment 40, hour 39, hr 92, latitude 92, location 92, midday 92, mod 15, solar_longitude 97, start 47, sunrise 92, sunset 92, tangent_degrees 92, and universal_from_standard 92.

96 (*astronomical lunar tests 96*) \equiv

```
def testUniversalFromDynamical(self):
    # from Meeus Example 10.a, pag 78
    date = gregorian_date(1977, FEBRUARY, 18)
    time = time_from_clock([3, 37, 40])
    td = fixed_from_gregorian(date) + time
    utc = universal_from_dynamical(td)
    clk = clock_from_moment(utc)
    self.assertEqual(hour(clk), 3)
    self.assertEqual(minute(clk), 36)
    self.assertEqual(isecond(round(seconds(clk))), 52)

def testDynamicalFromUniversal(self):
    # from Meeus Example 10.a, pag 78 (well, inverse of)
    date = gregorian_date(1977, FEBRUARY, 18)
    time = time_from_clock([3, 36, 52])
    utc = fixed_from_gregorian(date) + time
    td = dynamical_from_universal(utc)
    clk = clock_from_moment(td)
    self.assertEqual(hour(clk), 3)
    self.assertEqual(minute(clk), 37)
    self.assertEqual(isecond(round(seconds(clk))), 40)
    # from Meeus Example 10.b, pag 79
    # I should get 7:42 but I get [7, 57, mpf('54.660540372133255')]
    # The equivalent CL
    #   (load "calendrica-3.0.cl")
    #   (in-package "CC3")
    #   (setq date (gregorian-date 333 february 6))
    #   (setq time (time-from-clock '(6 0 0)))
    #   (setq utc (+ (fixed-from-gregorian date) time))
    #   (setq td (dynamical-from-universal utc))
    #   (setq clk (clock-from-moment td))
    # gives (7 57 54.660540566742383817L0) on CLisp on PC
    # The reply from Prof Reingold and Dershowitz says:
    # From      Ed Reingold <reingold@emr.cs.iit.edu>
    # To        Enrico Spinielli <enrico.spinielli@gmail.com>
    # Cc        nachumd@tau.ac.il
    # date      Thu, Aug 6, 2009 at 3:46 PM
    # subject   Re: dynamical-from-universal values differ from Meeus
    # mailed-by emr.cs.iit.edu
    # hide details Aug 6
    # Our value of the ephemeris correction closely matches the value
    # given on the NASA web site
    # http://eclipse.gsfc.nasa.gov/SEhelp/deltat2004.html for 333
    # (interpolating between the years 300 and 400), namely,
    # their value is 7027 seconds, while ours is 7075 seconds.
    # Meeus uses 6146 seconds, the difference amounts to about 14 minutes.
    # With Allegro Common Lisp, our functions
    #
    # (clock-from-moment (dynamical-from-universal
    #                     (+ (fixed-from-julian '(333 2 6)) 0.25L0)))
    #
    # give
    #
    #   (7 57 54.660540372133255d0)
    #
    # while CLisp on my PC gives
    #
    #   (7 57 54.660540566742383817L0)
    #
    # The difference in Delta-T explains Meeus's value of 7:42am.
```

```

#
# I then follow Calendrica Calculations (and NASA)
date = gregorian_date(333, FEBRUARY, 6)
time = time_from_clock([6, 0, 0])
utc = fixed_from_gregorian(date) + time
td = dynamical_from_universal(utc)
clk = clock_from_moment(td)
self.assertEqual(hour(clk), 7)
self.assertEqual(minute(clk), 57)
self.assertAlmostEqual(seconds(clk), 54.66054, 4)

```

This definition is continued in chunks 98, 100, 102, 104, 106, and 108.

This code is used in chunk 135.

Defines:

```

testDynamicalFromUniversal, never used.
testUniversalFromDynamical, never used.

```

Uses clock_from_moment 40, dynamical_from_universal 95, FEBRUARY 53, fixed_from_gregorian 55, gregorian_date 52, hour 39, iround 15, minute 39, seconds 39, time_from_clock 43, and universal_from_dynamical 95.


```

97  <astronomical lunar calendars 95>)+≡
    # see lines 3111-3114 in calendrica-3.0.cl
    J2000 = hr(mpf(12)) + gregorian_new_year(2000)

    # see lines 3116-3126 in calendrica-3.0.cl
    def sidereal_from_moment(tee):
        """Return the mean sidereal time of day from moment tee expressed
        as hour angle. Adapted from "Astronomical Algorithms"
        by Jean Meeus, Willmann_Bell, Inc., 1991."""
        c = (tee - J2000) / mpf(36525)
        return mod(poly(c, deg([mpf(280.46061837),
                                mpf(36525) * mpf(360.98564736629),
                                mpf(0.000387933),
                                mpf(-1)/mpf(38710000)])),
                    360)

    # see lines 3128-3130 in calendrica-3.0.cl
    MEAN_TROPICAL_YEAR = mpf(365.242189)

    # see lines 3132-3134 in calendrica-3.0.cl
    MEAN_SIDEREAL_YEAR = mpf(365.25636)

    # see lines 93-97 in calendrica-3.0.errata.cl
    MEAN_SYNODIC_MONTH = mpf(29.530588861)

    # see lines 3140-3176 in calendrica-3.0.cl
    def ephemeris_correction(tee):
        """Return Dynamical Time minus Universal Time (in days) for
        moment, tee. Adapted from "Astronomical Algorithms"
        by Jean Meeus, Willmann_Bell, Inc., 1991."""
        year = gregorian_year_from_fixed(ifloor(tee))
        c = gregorian_date_difference(gregorian_date(1900, JANUARY, 1),
                                     gregorian_date(year, JULY, 1)) / mpf(36525)

        if (1988 <= year <= 2019):
            return 1/86400 * (year - 1933)
        elif (1900 <= year <= 1987):
            return poly(c, [mpf(-0.00002), mpf(0.000297), mpf(0.025184),
                            mpf(-0.181133), mpf(0.553040), mpf(-0.861938),
                            mpf(0.677066), mpf(-0.212591)])
        elif (1800 <= year <= 1899):
            return poly(c, [mpf(-0.000009), mpf(0.003844), mpf(0.083563),
                            mpf(0.865736), mpf(4.867575), mpf(15.845535),
                            mpf(31.332267), mpf(38.291999), mpf(28.316289),
                            mpf(11.636204), mpf(2.043794)])
        elif (1700 <= year <= 1799):
            return (1/86400 *
                    poly(year - 1700, [8.118780842, -0.005092142,
                                         0.003336121, -0.0000266484]))
        elif (1620 <= year <= 1699):
            return (1/86400 *
                    poly(year - 1600,
                        [mpf(196.58333), mpf(-4.0675), mpf(0.0219167)]))
        else:
            x = (hr(mpf(12)) +
                 gregorian_date_difference(gregorian_date(1810, JANUARY, 1),
                                           gregorian_date(year, JANUARY, 1)))
            return 1/86400 * (((x * x) / mpf(41048480)) - 15)

    # see lines 3178-3207 in calendrica-3.0.cl
    def equation_of_time(tee):
        """Return the equation of time (as fraction of day) for moment, tee.

```

```

Adapted from "Astronomical Algorithms" by Jean Meeus,
Willmann_Bell, Inc., 1991.
c = julian_centuries(tee)
lamb = poly(c, deg([mpf(280.46645), mpf(36000.76983), mpf(0.0003032)]))
anomaly = poly(c, deg([mpf(357.52910), mpf(35999.05030),
    mpf(-0.0001559), mpf(-0.00000048)]))
eccentricity = poly(c, [mpf(0.016708617),
    mpf(-0.000042037),
    mpf(-0.0000001236)])
varepsilon = obliquity(tee)
y = pow(tangent_degrees(varepsilon / 2), 2)
equation = ((1/2 / pi) *
    (y * sin_degrees(2 * lamb) +
    -2 * eccentricity * sin_degrees(anomaly) +
    (4 * eccentricity * y * sin_degrees(anomaly) *
    cosine_degrees(2 * lamb)) +
    -0.5 * y * y * sin_degrees(4 * lamb) +
    -1.25 * eccentricity * eccentricity * sin_degrees(2 * anomaly)))
return signum(equation) * min(abs(equation), hr(mpf(12)))

# see lines 3209-3259 in calendrica-3.0.cl
def solar_longitude(tee):
    """Return the longitude of sun at moment 'tee'.
    Adapted from 'Planetary Programs and Tables from -4000 to +2800'
    by Pierre Bretagnon and Jean-Louis Simon, Willmann_Bell, Inc., 1986.
    See also pag 166 of 'Astronomical Algorithms' by Jean Meeus, 2nd Ed 1998,
    with corrections Jun 2005."""
    c = julian_centuries(tee)
    coefficients = [403406, 195207, 119433, 112392, 3891, 2819, 1721,
        660, 350, 334, 314, 268, 242, 234, 158, 132, 129, 114,
        99, 93, 86, 78, 72, 68, 64, 46, 38, 37, 32, 29, 28, 27, 27,
        25, 24, 21, 21, 20, 18, 17, 14, 13, 13, 13, 12, 10, 10, 10,
        10]
    multipliers = [mpf(0.9287892), mpf(35999.1376958), mpf(35999.4089666),
        mpf(35998.7287385), mpf(71998.20261), mpf(71998.4403),
        mpf(36000.35726), mpf(71997.4812), mpf(32964.4678),
        mpf(-19.4410), mpf(445267.1117), mpf(45036.8840), mpf(3.1008),
        mpf(22518.4434), mpf(-19.9739), mpf(65928.9345),
        mpf(9038.0293), mpf(3034.7684), mpf(33718.148), mpf(3034.448),
        mpf(-2280.773), mpf(29929.992), mpf(31556.493), mpf(149.588),
        mpf(9037.750), mpf(107997.405), mpf(-4444.176), mpf(151.771),
        mpf(67555.316), mpf(31556.080), mpf(-4561.540),
        mpf(107996.706), mpf(1221.655), mpf(62894.167),
        mpf(31437.369), mpf(14578.298), mpf(-31931.757),
        mpf(34777.243), mpf(1221.999), mpf(62894.511),
        mpf(-4442.039), mpf(107997.909), mpf(119.066), mpf(16859.071),
        mpf(-4.578), mpf(26895.292), mpf(-39.127), mpf(12297.536),
        mpf(90073.778)]
    addends = [mpf(270.54861), mpf(340.19128), mpf(63.91854), mpf(331.26220),
        mpf(317.843), mpf(86.631), mpf(240.052), mpf(310.26), mpf(247.23),
        mpf(260.87), mpf(297.82), mpf(343.14), mpf(166.79), mpf(81.53),
        mpf(3.50), mpf(132.75), mpf(182.95), mpf(162.03), mpf(29.8),
        mpf(266.4), mpf(249.2), mpf(157.6), mpf(257.8), mpf(185.1),
        mpf(69.9), mpf(8.0), mpf(197.1), mpf(250.4), mpf(65.3),
        mpf(162.7), mpf(341.5), mpf(291.6), mpf(98.5), mpf(146.7),
        mpf(110.0), mpf(5.2), mpf(342.6), mpf(230.9), mpf(256.1),
        mpf(45.3), mpf(242.9), mpf(115.2), mpf(151.8), mpf(285.3),
        mpf(53.3), mpf(126.6), mpf(205.7), mpf(85.9), mpf(146.1)]
    lam = (deg(mpf(282.7771834)) +
        deg(mpf(36000.76953744)) * c +
        deg(mpf(0.000005729577951308232)) *

```

```

        sigma([coefficients, addends, multipliers],
              lambda x, y, z: x * sin_degrees(y + (z * c))))
    return mod(lam + aberration(tee) + nutation(tee), 360)

# see lines 3261-3271 in calendrica-3.0.cl
def nutation(tee):
    """Return the longitudinal nutation at moment, tee."""
    c = julian_centuries(tee)
    cap_A = poly(c, deg([mpf(124.90), mpf(-1934.134), mpf(0.002063)]))
    cap_B = poly(c, deg([mpf(201.11), mpf(72001.5377), mpf(0.00057)]))
    return (deg(mpf(-0.004778)) * sin_degrees(cap_A) +
            deg(mpf(-0.0003667)) * sin_degrees(cap_B))

```

This code is used in chunk 3.

Defines:

ephemeris_correction, used in chunk 95.
 equation_of_time, used in chunk 92.
 J2000, used in chunks 92, 99, and 109.
 MEAN_SIDEREAL_YEAR, used in chunk 126.
 MEAN_SYNODIC_MONTH, used in chunks 109, 123, and 132.
 MEAN_TROPICAL_YEAR, used in chunks 99, 112, 115, 120, and 123.
 nutation, used in chunks 98 and 109.
 sidereal_from_moment, used in chunk 110.
 solar_longitude, used in chunks 92, 95, 99, 109, 112, 115, 120, 123, 126, and 186.

Uses aberration 99, angle 92, cosine_degrees 92, deg 92, gregorian_date 52,
 gregorian_date_difference 56, gregorian_new_year 56, gregorian_year_from_fixed 55, hour 39,
 hr 92, ifloor 15, JANUARY 53, julian_centuries 92, JULY 53, longitude 92, mod 15, obliquity 92,
 poly 34, sigma 31, signum 92, sin_degrees 92, and tangent_degrees 92.

```

98  <astronomical lunar tests 96>+≡
    def testNutation(self):
        # from Meeus, pag 343
        TD = fixed_from_gregorian(gregorian_date(1992, APRIL, 12))
        tee = universal_from_dynamical(TD)
        self.assertAlmostEqual(nutation(tee), mpf(0.004610), 3)

```

This code is used in chunk 135.

Defines:

testNutation, never used.

Uses APRIL 53, fixed_from_gregorian 55, gregorian_date 52, nutation 97,
 and universal_from_dynamical 95.

```

99  <astronomical lunar calendars 95>+≡
    # see lines 3273-3281 in calendrica-3.0.cl
    def aberration(tee):
        """Return the aberration at moment, tee."""
        c = julian_centuries(tee)
        return ((deg(mpf(0.0000974)) *
                 cosine_degrees(deg(mpf(177.63)) + deg(mpf(35999.01848)) * c)) -
                 deg(mpf(0.005575)))

    # see lines 3283-3295 in calendrica-3.0.cl
    def solar_longitude_after(lam, tee):
        """Return the moment UT of the first time at or after moment, tee,
        when the solar longitude will be lam degrees."""
        rate = MEAN_TROPICAL_YEAR / deg(360)
        tau = tee + rate * mod(lam - solar_longitude(tee), 360)
        a = max(tee, tau - 5)
        b = tau + 5
        return invertAngular(solar_longitude, lam, a, b)

    # see lines 3297-3300 in calendrica-3.0.cl
    SPRING = deg(0)

    # see lines 3302-3305 in calendrica-3.0.cl
    SUMMER = deg(90)

    # see lines 3307-3310 in calendrica-3.0.cl
    AUTUMN = deg(180)

    # see lines 3312-3315 in calendrica-3.0.cl
    WINTER = deg(270)

    # see lines 3317-3339 in calendrica-3.0.cl
    def precession(tee):
        """Return the precession at moment tee using 0,0 as J2000 coordinates.
        Adapted from "Astronomical Algorithms" by Jean Meeus,
        Willmann-Bell, Inc., 1991."""
        c = julian_centuries(tee)
        eta = mod(poly(c, [0,
                           secs(mpf(47.0029)),
                           secs(mpf(-0.03302)),
                           secs(mpf(0.000060))]),
                  360)
        cap_P = mod(poly(c, [deg(mpf(174.876384)),
                             secs(mpf(-869.8089)),
                             secs(mpf(0.03536))]),
                    360)
        p = mod(poly(c, [0,
                         secs(mpf(5029.0966)),
                         secs(mpf(1.11113)),
                         secs(mpf(0.000006))]),
                 360)
        cap_A = cosine_degrees(eta) * sin_degrees(cap_P)
        cap_B = cosine_degrees(cap_P)
        arg = arctan_degrees(cap_A, cap_B)

        return mod(p + cap_P - arg, 360)

    # see lines 3341-3347 in calendrica-3.0.cl
    def sidereal_solar_longitude(tee):
        """Return sidereal solar longitude at moment, tee."""
        return mod(solar_longitude(tee) - precession(tee) + SIDEREAL_START, 360)

```

```

# see lines 3349-3365 in calendrica-3.0.cl
def estimate_prior_solar_longitude(lam, tee):
    """Return approximate moment at or before tee
    when solar longitude just exceeded lam degrees."""
    rate = MEAN_TROPICAL_YEAR / deg(360)
    tau = tee - (rate * mod(solar_longitude(tee) - lam, 360))
    cap_Delta = mod(solar_longitude(tau) - lam + deg(180), 360) - deg(180)
    return min(tee, tau - (rate * cap_Delta))

# see lines 3367-3376 in calendrica-3.0.cl
def mean_lunar_longitude(c):
    """Return mean longitude of moon (in degrees) at moment
    given in Julian centuries c (including the constant term of the
    effect of the light-time (-0".70).
    Adapted from eq. 47.1 in "Astronomical Algorithms" by Jean Meeus,
    Willmann_Bell, Inc., 2nd ed. with corrections, 2005."""
    return degrees(poly(c, deg([mpf(218.3164477), mpf(481267.88123421),
                                mpf(-0.0015786), mpf(1/538841),
                                mpf(-1/65194000)])))

```

This code is used in chunk 3.

Defines:

aberration, used in chunk 97.
 AUTUMN, used in chunks 120 and 186.
 estimate_prior_solar_longitude, used in chunks 112, 115, 120, and 123.
 mean_lunar_longitude, used in chunks 100 and 109.
 precession, used in chunks 109 and 126.
 sidereal_solar_longitude, used in chunk 126.
 solar_longitude_after, used in chunks 92, 123, 132, and 186.
 SPRING, used in chunks 112, 115, 132, and 186.
 SUMMER, used in chunk 186.
 WINTER, used in chunks 92, 123, and 186.

Uses arctan_degrees 92 92, cosine_degrees 92, deg 92, degrees 92, invert_angular 28 28,
 J2000 97, julian_centuries 92, longitude 92, MEAN_TROPICAL_YEAR 97, mod 15, poly 34, secs 92,
 SIDEREAL_START 126, sin_degrees 92, and solar_longitude 97.

```

100 <astronomical lunar tests 96>+≡
    def testMeanLunarLongitude(self):
        # from Example 47.a in Jan Meeus "Astronomical Algorithms" pag 342
        self.assertAlmostEqual(mean_lunar_longitude(-0.077221081451), 134.290182, 6)

```

This code is used in chunk 135.

Defines:

testMeanLunarLongitude, never used.

Uses mean_lunar_longitude 99.

```

101 <astronomical lunar calendars 95>+≡
    # see lines 3378-3387 in calendrica-3.0.cl
    def lunar_elongation(c):
        """Return elongation of moon (in degrees) at moment
        given in Julian centuries c.
        Adapted from eq. 47.2 in "Astronomical Algorithms" by Jean Meeus,
        Willmann_Bell, Inc., 2nd ed. with corrections, 2005."""
        return degrees(poly(c, deg([mpf(297.8501921), mpf(445267.1114034),
                                    mpf(-0.0018819), mpf(1/545868),
                                    mpf(-1/113065000)])))

```

This code is used in chunk 3.

Defines:

lunar_elongation, used in chunks 102, 109, and 110.

Uses deg 92, degrees 92, and poly 34.

```

102  <astronomical lunar tests 96>+≡
      def testLunarElongation(self):
          # from Example 47.a in Jan Meeus "Astronomical Algorithms" pag 342
          self.assertAlmostEqual(lunar_elongation(-0.077221081451), 113.842304, 6)

```

This code is used in chunk 135.

Defines:

testLunarElongation, never used.

Uses lunar_elongation 101.

```

103  <astronomical lunar calendars 95>+≡
      # see lines 3389-3398 in calendrica-3.0.cl
      def solar_anomaly(c):
          """Return mean anomaly of sun (in degrees) at moment
          given in Julian centuries c.
          Adapted from eq. 47.3 in "Astronomical Algorithms" by Jean Meeus,
          Willmann_Bell, Inc., 2nd ed. with corrections, 2005."""
          return degrees(poly(c, deg([mpf(357.5291092), mpf(35999.0502909),
                                     mpf(-0.0001536), mpf(1/24490000)])))

```

This code is used in chunk 3.

Defines:

solar_anomaly, used in chunks 104, 109, 110, and 129.

Uses deg 92, degrees 92, and poly 34.

```

104  <astronomical lunar tests 96>+≡
      def testSolarAnomaly(self):
          # from Example 47.a in Jan Meeus "Astronomical Algorithms" pag 342
          self.assertAlmostEqual(solar_anomaly(-0.077221081451), 97.643514, 6)

```

This code is used in chunk 135.

Defines:

testSolarAnomaly, never used.

Uses solar_anomaly 103.

```

105  <astronomical lunar calendars 95>+≡
      # see lines 3400-3409 in calendrica-3.0.cl
      def lunar_anomaly(c):
          """Return mean anomaly of moon (in degrees) at moment
          given in Julian centuries c.
          Adapted from eq. 47.4 in "Astronomical Algorithms" by Jean Meeus,
          Willmann_Bell, Inc., 2nd ed. with corrections, 2005."""
          return degrees(poly(c, deg([mpf(134.9633964), mpf(477198.8675055),
                                     mpf(0.0087414), mpf(1/69699),
                                     mpf(-1/14712000)])))

```

This code is used in chunk 3.

Defines:

lunar_anomaly, used in chunks 106, 109, 110, and 129.

Uses deg 92, degrees 92, and poly 34.

```

106  <astronomical lunar tests 96>+≡
      def testLunarAnomaly(self):
          # from Example 47.a in Jan Meeus "Astronomical Algorithms" pag 342
          self.assertAlmostEqual(lunar_anomaly(-0.077221081451), 5.150833, 6)

```

This code is used in chunk 135.

Defines:

testLunarAnomaly, never used.

Uses lunar_anomaly 105.

```

107  <astronomical lunar calendars 95>+≡
      # see lines 3411-3420 in calendrica-3.0.cl
      def moon_node(c):
          """Return Moon's argument of latitude (in degrees) at moment
          given in Julian centuries 'c'.
          Adapted from eq. 47.5 in "Astronomical Algorithms" by Jean Meeus,
          Willmann_Bell, Inc., 2nd ed. with corrections, 2005."""
          return degrees(poly(c, deg([mpf(93.2720950), mpf(483202.0175233),
                                     mpf(-0.0036539), mpf(-1/3526000),
                                     mpf(1/863310000)])))

```

This code is used in chunk 3.

Defines:

 moon_node, used in chunks 108-110.

Uses deg 92, degrees 92, latitude 92, and poly 34.

```

108  <astronomical lunar tests 96>+≡
      def testMoonNode(self):
          # from Example 47.a in Jan Meeus "Astronomical Algorithms" pag 342
          self.assertAlmostEqual(moon_node(-0.077221081451), 219.889721, 6)

```

This code is used in chunk 135.

Defines:

 testMoonNode, never used.

Uses moon_node 107.

```

109  <astronomical lunar calendars 95>+≡
# see lines 3422-3485 in calendrica-3.0.cl
def lunar_longitude(tee):
    """Return longitude of moon (in degrees) at moment tee.
    Adapted from "Astronomical Algorithms" by Jean Meeus,
    Willmann_Bell, Inc., 2nd ed., 1998."""
    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)
    # see eq. 47.6 in Meeus
    cap_E = poly(c, [1, mpf(-0.002516), mpf(-0.0000074)])
    args_lunar_elongation = \
        [0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 0, 0, 4, 0, 4, 2, 2, 1,
         1, 2, 2, 4, 2, 0, 2, 2, 1, 2, 0, 0, 2, 2, 2, 4, 0, 3, 2, 4, 0, 2,
         2, 2, 4, 0, 4, 1, 2, 0, 1, 3, 4, 2, 0, 1, 2]
    args_solar_anomaly = \
        [0, 0, 0, 0, 1, 0, 0, -1, 0, -1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1,
         0, 1, -1, 0, 0, 0, 1, 0, -1, 0, -2, 1, 2, -2, 0, 0, -1, 0, 0, 1,
         -1, 2, 2, 1, -1, 0, 0, -1, 0, 1, 0, 1, 0, 0, -1, 2, 1, 0]
    args_lunar_anomaly = \
        [1, -1, 0, 2, 0, 0, -2, -1, 1, 0, -1, 0, 1, 0, 1, 1, -1, 3, -2,
         -1, 0, -1, 0, 1, 2, 0, -3, -2, -1, -2, 1, 0, 2, 0, -1, 1, 0,
         -1, 2, -1, 1, -2, -1, -1, -2, 0, 1, 4, 0, -2, 0, 2, 1, -2, -3,
         2, 1, -1, 3]
    args_moon_node = \
        [0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, -2, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, -2, 2, 0, 2, 0, 0, 0, 0,
         0, 0, -2, 0, 0, 0, 0, -2, -2, 0, 0, 0, 0, 0, 0, 0, 0]
    sine_coefficients = \
        [6288774, 1274027, 658314, 213618, -185116, -114332,
         58793, 57066, 53322, 45758, -40923, -34720, -30383,
         15327, -12528, 10980, 10675, 10034, 8548, -7888,
         -6766, -5163, 4987, 4036, 3994, 3861, 3665, -2689,
         -2602, 2390, -2348, 2236, -2120, -2069, 2048, -1773,
         -1595, 1215, -1110, -892, -810, 759, -713, -700, 691,
         596, 549, 537, 520, -487, -399, -381, 351, -340, 330,
         327, -323, 299, 294]
    correction = (deg(1/1000000) *
        sigma([sine_coefficients, args_lunar_elongation,
               args_solar_anomaly, args_lunar_anomaly,
               args_moon_node],
        lambda v, w, x, y, z:
        v * pow(cap_E, abs(x)) *
        sin_degrees((w * cap_D) +
                     (x * cap_M) +
                     (y * cap_M_prime) +
                     (z * cap_F))))
    A1 = deg(mpf(119.75)) + (c * deg(mpf(131.849)))
    venus = (deg(3958/1000000) * sin_degrees(A1))
    A2 = deg(mpf(53.09)) + c * deg(mpf(479264.29))
    jupiter = (deg(318/1000000) * sin_degrees(A2))
    flat_earth = (deg(1962/1000000) * sin_degrees(cap_L_prime - cap_F))

    return mod(cap_L_prime + correction + venus +
               jupiter + flat_earth + nutation(tee), 360)

# see lines 3663-3732 in calendrica-3.0.cl
def lunar_latitude(tee):

```



```

"""Return the latitude of moon (in degrees) at moment, tee.
Adapted from "Astronomical Algorithms" by Jean Meeus,
Willmann_Bell, Inc., 1998."""
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, [1, mpf(-0.002516), mpf(-0.0000074)])
args_lunar_elongation = \
    [0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 2, 2, 2, 2, 2, 0, 4, 0, 0, 0,
     1, 0, 0, 0, 1, 0, 4, 4, 0, 4, 2, 2, 2, 2, 0, 2, 2, 2, 2, 4, 2, 2,
     0, 2, 1, 1, 0, 2, 1, 2, 0, 4, 4, 1, 4, 1, 4, 2]
args_solar_anomaly = \
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0, 1, -1, -1, -1, 1, 0, 1,
     0, 1, 0, 1, 1, 1, 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 = \
    [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, -2,
     0, 0]
args_moon_node = \
    [1, 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, 3, -1, -1, 1, -1, -1, 1, -1, 1, -1,
     -1, -1, -1, -1, -1, 1]
sine_coefficients = \
    [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 = (deg(1/1000000) *
        sigma([sine_coefficients,
                 args_lunar_elongation,
                 args_solar_anomaly,
                 args_lunar_anomaly,
                 args_moon_node],
               lambda v, w, x, y, z: (v *
                                       pow(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(mpf(119.75))) + c * deg(mpf(131.849))) + cap_F) +
         (sin_degrees(deg(mpf(119.75))) + c * deg(mpf(131.849))) - cap_F))
flat_earth = (deg(-2235/1000000) * sin_degrees(cap_L_prime) +
              deg(127/1000000) * sin_degrees(cap_L_prime - cap_M_prime) +
              deg(-115/1000000) * sin_degrees(cap_L_prime + cap_M_prime))
extra = (deg(382/1000000) *
         (sin_degrees(deg(mpf(313.45))) + c * deg(mpf(481266.484))))
return beta + venus + flat_earth + extra

# see lines 192-197 in calendrica-3.0.errata.cl
def lunar_node(tee):

```

```

    """Return Angular distance of the node from the equinoctial point
    at fixed moment, tee.
    Adapted from eq. 47.7 in "Astronomical Algorithms"
    by Jean Meeus, Willmann_Bell, Inc., 2nd ed., 1998
    with corrections June 2005."""
    return mod(moon_node(julian_centuries(tee)) + deg(90), 180) - 90

def alt_lunar_node(tee):
    """Return Angular distance of the node from the equinoctial point
    at fixed moment, tee.
    Adapted from eq. 47.7 in "Astronomical Algorithms"
    by Jean Meeus, Willmann_Bell, Inc., 2nd ed., 1998
    with corrections June 2005."""
    return degrees(poly(julian_centuries(tee), deg([mpf(125.0445479),
                                                    mpf(-1934.1362891),
                                                    mpf(0.0020754),
                                                    mpf(1/467441),
                                                    mpf(-1/60616000)]))))

def lunar_true_node(tee):
    """Return Angular distance of the true node (the node of the instantaneous
    lunar orbit) from the equinoctial point at moment, tee.
    Adapted from eq. 47.7 and pag. 344 in "Astronomical Algorithms"
    by Jean Meeus, Willmann_Bell, Inc., 2nd ed., 1998
    with corrections June 2005."""
    c = julian_centuries(tee)
    cap_D = lunar_elongation(c)
    cap_M = solar_anomaly(c)
    cap_M_prime = lunar_anomaly(c)
    cap_F = moon_node(c)
    periodic_terms = (deg(-1.4979) * sin_degrees(2 * (cap_D - cap_F)) +
                      deg(-0.1500) * sin_degrees(cap_M) +
                      deg(-0.1226) * sin_degrees(2 * cap_D) +
                      deg(0.1176) * sin_degrees(2 * cap_F) +
                      deg(-0.0801) * sin_degrees(2 * (cap_M_prime - cap_F)))
    return alt_lunar_node(tee) + periodic_terms

def lunar_perigee(tee):
    """Return Angular distance of the perigee from the equinoctial point
    at moment, tee.
    Adapted from eq. 47.7 in "Astronomical Algorithms"
    by Jean Meeus, Willmann_Bell, Inc., 2nd ed., 1998
    with corrections June 2005."""
    return degrees(poly(julian_centuries(tee), deg([mpf(83.3532465),
                                                    mpf(4069.0137287),
                                                    mpf(-0.0103200),
                                                    mpf(-1/80053),
                                                    mpf(1/18999000)]))))

# see lines 199-206 in calendrica-3.0.errata.cl
def sidereal_lunar_longitude(tee):
    """Return sidereal lunar longitude at moment, tee."""
    return mod(lunar_longitude(tee) - precession(tee) + SIDEREAL_START, 360)

# see lines 99-190 in calendrica-3.0.errata.cl
def nth_new_moon(n):
    """Return the moment of n-th new moon after (or before) the new moon
    of January 11, 1. Adapted from "Astronomical Algorithms"
    by Jean Meeus, Willmann_Bell, Inc., 2nd ed., 1998."""

```

```

n0 = 24724
k = n - n0
c = k / mpf(1236.85)
approx = (J2000 +
    poly(c, [mpf(5.09766),
        MEAN_SYNODIC_MONTH * mpf(1236.85),
        mpf(0.0001437),
        mpf(-0.000000150),
        mpf(0.00000000073)]))
cap_E = poly(c, [1, mpf(-0.002516), mpf(-0.0000074)])
solar_anomaly = poly(c, deg([mpf(2.5534),
    (mpf(1236.85) * mpf(29.10535669)),
    mpf(-0.0000014), mpf(-0.00000011)]))
lunar_anomaly = poly(c, deg([mpf(201.5643),
    (mpf(385.81693528) * mpf(1236.85)),
    mpf(0.0107582), mpf(0.00001238),
    mpf(-0.000000058)]))
moon_argument = poly(c, deg([mpf(160.7108),
    (mpf(390.67050284) * mpf(1236.85)),
    mpf(-0.0016118), mpf(-0.00000227),
    mpf(0.000000011)]))
cap_omega = poly(c, [mpf(124.7746),
    (mpf(-1.56375588) * mpf(1236.85)),
    mpf(0.0020672), mpf(0.00000215)])
E_factor = [0, 1, 0, 0, 1, 1, 2, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0]
solar_coeff = [0, 1, 0, 0, -1, 1, 2, 0, 0, 1, 0, 1, 1, -1, 2,
    0, 3, 1, 0, 1, -1, -1, 1, 0]
lunar_coeff = [1, 0, 2, 0, 1, 1, 0, 1, 1, 2, 3, 0, 0, 2, 1, 2,
    0, 1, 2, 1, 1, 1, 3, 4]
moon_coeff = [0, 0, 0, 2, 0, 0, 0, -2, 2, 0, 0, 2, -2, 0, 0,
    -2, 0, -2, 2, 2, 2, -2, 0, 0]
sine_coeff = [mpf(-0.40720), mpf(0.17241), mpf(0.01608),
    mpf(0.01039), mpf(0.00739), mpf(-0.00514),
    mpf(0.00208), mpf(-0.00111), mpf(-0.00057),
    mpf(0.00056), mpf(-0.00042), mpf(0.00042),
    mpf(0.00038), mpf(-0.00024), mpf(-0.00007),
    mpf(0.00004), mpf(0.00004), mpf(0.00003),
    mpf(0.00003), mpf(-0.00003), mpf(0.00003),
    mpf(-0.00002), mpf(-0.00002), mpf(0.00002)]
correction = ((deg(mpf(-0.00017)) * sin_degrees(cap_omega)) +
    sigma([sine_coeff, E_factor, solar_coeff,
        lunar_coeff, moon_coeff],
        lambda v, w, x, y, z: (v *
            pow(cap_E, w) *
                sin_degrees((x * solar_anomaly) +
                    (y * lunar_anomaly) +
                    (z * moon_argument))))))
add_const = [mpf(251.88), mpf(251.83), mpf(349.42), mpf(84.66),
    mpf(141.74), mpf(207.14), mpf(154.84), mpf(34.52),
    mpf(207.19), mpf(291.34), mpf(161.72), mpf(239.56),
    mpf(331.55)]
add_coeff = [mpf(0.016321), mpf(26.651886), mpf(36.412478),
    mpf(18.206239), mpf(53.303771), mpf(2.453732),
    mpf(7.306860), mpf(27.261239), mpf(0.121824),
    mpf(1.844379), mpf(24.198154), mpf(25.513099),
    mpf(3.592518)]
add_factor = [mpf(0.000165), mpf(0.000164), mpf(0.000126),
    mpf(0.000110), mpf(0.000062), mpf(0.000060),
    mpf(0.000056), mpf(0.000047), mpf(0.000042),
    mpf(0.000040), mpf(0.000037), mpf(0.000035),

```

```

        mpf(0.000023)]
    extra = (deg(mpf(0.000325)) *
             sin_degrees(poly(c, deg([mpf(299.77), mpf(132.8475848),
                                     mpf(-0.009173)]))))
    additional = sigma([add_const, add_coeff, add_factor],
                       lambda i, j, l: l * sin_degrees(i + j * k))

    return universal_from_dynamical(approx + correction + extra + additional)

# see lines 3578-3585 in calendrica-3.0.cl
def new_moon_before(tee):
    """Return the moment UT of last new moon before moment tee."""
    t0 = nth_new_moon(0)
    phi = lunar_phase(tee)
    n = iround(((tee - t0) / MEAN_SYNODIC_MONTH) - (phi / deg(360)))
    return nth_new_moon(final(n - 1, lambda k: nth_new_moon(k) < tee))

# see lines 3587-3594 in calendrica-3.0.cl
def new_moon_at_or_after(tee):
    """Return the moment UT of first new moon at or after moment, tee."""
    t0 = nth_new_moon(0)
    phi = lunar_phase(tee)
    n = iround((tee - t0) / MEAN_SYNODIC_MONTH - phi / deg(360))
    return nth_new_moon(next(n, lambda k: nth_new_moon(k) >= tee))

# see lines 3596-3613 in calendrica-3.0.cl
def lunar_phase(tee):
    """Return the lunar phase, as an angle in degrees, at moment tee.
    An angle of 0 means a new moon, 90 degrees means the
    first quarter, 180 means a full moon, and 270 degrees
    means the last quarter."""
    phi = mod(lunar_longitude(tee) - solar_longitude(tee), 360)
    t0 = nth_new_moon(0)
    n = iround((tee - t0) / MEAN_SYNODIC_MONTH)
    phi_prime = (deg(360) *
                 mod((tee - nth_new_moon(n)) / MEAN_SYNODIC_MONTH, 1))
    if abs(phi - phi_prime) > deg(180):
        return phi_prime
    else:
        return phi

# see lines 3615-3625 in calendrica-3.0.cl
def lunar_phase_at_or_before(phi, tee):
    """Return the moment UT of the last time at or before moment, tee,
    when the lunar_phase was phi degrees."""
    tau = (tee -
           (MEAN_SYNODIC_MONTH *
            (1/deg(360)) *
            mod(lunar_phase(tee) - phi, 360)))
    a = tau - 2
    b = min(tee, tau + 2)
    return invert_angular(lunar_phase, phi, a, b)

# see lines 3627-3631 in calendrica-3.0.cl
NEW = deg(0)

```

```

# see lines 3633-3637 in calendrica-3.0.cl
FIRST_QUARTER = deg(90)

# see lines 3639-3643 in calendrica-3.0.cl
FULL = deg(180)

# see lines 3645-3649 in calendrica-3.0.cl
LAST_QUARTER = deg(270)

# see lines 3651-3661 in calendrica-3.0.cl
def lunar_phase_at_or_after(phi, tee):
    """Return the moment UT of the next time at or after moment, tee,
    when the lunar_phase is phi degrees."""
    tau = (tee +
           (MEAN_SYNODIC_MONTH *
            (1/deg(360)) *
            mod(phi - lunar_phase(tee), 360)))
    a = max(tee, tau - 2)
    b = tau + 2
    return invert_angular(lunar_phase, phi, a, b)

```

This code is used in chunk 3.

Defines:

- alt_lunar_node, never used.
- FIRST_QUARTER, used in chunk 132.
- FULL, used in chunk 132.
- LAST_QUARTER, never used.
- lunar_latitude, used in chunks 94, 110, and 132.
- lunar_longitude, used in chunks 94, 110, and 186.
- lunar_node, never used.
- lunar_perigee, never used.
- lunar_phase, used in chunks 92, 126, and 132.
- lunar_phase_at_or_after, used in chunk 132.
- lunar_phase_at_or_before, never used.
- lunar_true_node, never used.
- NEW, used in chunk 132.
- new_moon_at_or_after, used in chunks 123, 126, and 186.
- new_moon_before, used in chunks 123 and 126.
- nth_new_moon, never used.
- sidereal_lunar_longitude, never used.

Uses angle 92, deg 92, degrees 92, final 19, invert_angular 28 28, iround 15, J2000 97, julian_centuries 92, latitude 92, longitude 92, lunar_anomaly 105, lunar_elongation 101, mean_lunar_longitude 99, MEAN_SYNODIC_MONTH 97, mod 15, moon_node 107, next 16, nutation 97, poly 34, precession 99, SIDEREAL_START 126, sigma 31, sin_degrees 92, solar_anomaly 103, solar_longitude 97, and universal_from_dynamical 95.

```

110  (astronomical lunar calendars 95)+≡
    # see lines 3734-3762 in calendrica-3.0.c1
    def lunar_altitude(tee, location):
        """Return the geocentric altitude of moon at moment, tee,
        at location, location, as a small positive/negative angle in degrees,
        ignoring parallax and refraction. Adapted from 'Astronomical
        Algorithms' by Jean Meeus, Willmann_Bell, Inc., 1998."""
        phi = latitude(location)
        psi = longitude(location)
        lamb = lunar_longitude(tee)
        beta = lunar_latitude(tee)
        alpha = right_ascension(tee, beta, lamb)
        delta = declination(tee, beta, lamb)
        theta0 = sidereal_from_moment(tee)
        cap_H = mod(theta0 + psi - alpha, 360)
        altitude = arcsin_degrees(
            (sin_degrees(phi) * sin_degrees(delta)) +
            (cosine_degrees(phi) * cosine_degrees(delta) * cosine_degrees(cap_H)))
        return mod(altitude + deg(180), 360) - deg(180)

    # see lines 3764-3813 in calendrica-3.0.c1
    def lunar_distance(tee):
        """Return the distance to moon (in meters) at moment, tee.
        Adapted from "Astronomical Algorithms" by Jean Meeus,
        Willmann_Bell, Inc., 2nd ed."""
        c = julian_centuries(tee)
        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, [1, mpf(-0.002516), mpf(-0.0000074)])
        args_lunar_elongation = \
            [0, 2, 2, 0, 0, 0, 2, 2, 2, 2, 0, 1, 0, 2, 0, 0, 4, 0, 4, 2, 2, 1,
             1, 2, 2, 4, 2, 0, 2, 2, 1, 2, 0, 0, 2, 2, 2, 4, 0, 3, 2, 4, 0, 2,
             2, 2, 4, 0, 4, 1, 2, 0, 1, 3, 4, 2, 0, 1, 2, 2,]
        args_solar_anomaly = \
            [0, 0, 0, 0, 1, 0, 0, -1, 0, -1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
             0, 1, -1, 0, 0, 0, 1, 0, -1, 0, -2, 1, 2, -2, 0, 0, -1, 0, 0, 1,
             -1, 2, 2, 1, -1, 0, 0, -1, 0, 1, 0, 1, 0, 0, -1, 2, 1, 0, 0]
        args_lunar_anomaly = \
            [1, -1, 0, 2, 0, 0, -2, -1, 1, 0, -1, 0, 1, 0, 1, 1, -1, 3, -2,
             -1, 0, -1, 0, 1, 2, 0, -3, -2, -1, -2, 1, 0, 2, 0, -1, 1, 0,
             -1, 2, -1, 1, -2, -1, -1, -2, 0, 1, 4, 0, -2, 0, 2, 1, -2, -3,
             2, 1, -1, 3, -1]
        args_moon_node = \
            [0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, -2, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, -2, 2, 0, 2, 0, 0, 0, 0,
             0, 0, -2, 0, 0, 0, 0, -2, -2, 0, 0, 0, 0, 0, 0, 0, -2]
        cosine_coefficients = \
            [-20905355, -3699111, -2955968, -569925, 48888, -3149,
             246158, -152138, -170733, -204586, -129620, 108743,
             104755, 10321, 0, 79661, -34782, -23210, -21636, 24208,
             30824, -8379, -16675, -12831, -10445, -11650, 14403,
             -7003, 0, 10056, 6322, -9884, 5751, 0, -4950, 4130, 0,
             -3958, 0, 3258, 2616, -1897, -2117, 2354, 0, 0, -1423,
             -1117, -1571, -1739, 0, -4421, 0, 0, 0, 0, 1165, 0, 0,
             8752]
        correction = sigma ([cosine_coefficients,
                               args_lunar_elongation,
                               args_solar_anomaly,

```

```

        args_lunar_anomaly,
        args_moon_node],
    lambda v, w, x, y, z: (v *
        pow(cap_E, abs(x)) *
        cosine_degrees((w * cap_D) +
            (x * cap_M) +
            (y * cap_M_prime) +
            (z * cap_F))))

    return mt(385000560) + correction

def lunar_position(tee):
    """Return the moon position (geocentric latitude and longitude [in degrees]
    and distance [in meters]) at moment, tee.
    Adapted from "Astronomical Algorithms" by Jean Meeus,
    Willmann_Bell, Inc., 2nd ed."""
    return (lunar_latitude(tee), lunar_longitude(tee), lunar_distance(tee))

# see lines 3815-3824 in calendrica-3.0.c1
def lunar_parallax(tee, location):
    """Return the parallax of moon at moment, tee, at location, location.
    Adapted from "Astronomical Algorithms" by Jean Meeus,
    Willmann_Bell, Inc., 1998."""
    geo = lunar_altitude(tee, location)
    Delta = lunar_distance(tee)
    alt = mt(6378140) / Delta
    arg = alt * cosine_degrees(geo)
    return arcsin_degrees(arg)

# see lines 3826-3832 in calendrica-3.0.c1
def topocentric_lunar_altitude(tee, location):
    """Return the topocentric altitude of moon at moment, tee,
    at location, location, as a small positive/negative angle in degrees,
    ignoring refraction."""
    return lunar_altitude(tee, location) - lunar_parallax(tee, location)

# see lines 3834-3839 in calendrica-3.0.c1
def lunar_diameter(tee):
    """Return the geocentric apparent lunar diameter of the moon (in
    degrees) at moment, tee. Adapted from 'Astronomical
    Algorithms' by Jean Meeus, Willmann_Bell, Inc., 2nd ed."""
    return deg(1792367000/9) / lunar_distance(tee)

```

This code is used in chunk 3.

Defines:

- lunar_altitude, used in chunks 94 and 132.
- lunar_diameter, never used.
- lunar_distance, never used.
- lunar_parallax, never used.
- lunar_position, never used.
- topocentric_lunar_altitude, used in chunk 92.

Uses angle 92, arcsin_degrees 92, cosine_degrees 92, declination 92, deg 92, degrees 92, julian_centuries 92, latitude 92, location 92, longitude 92, lunar_anomaly 105, lunar_elongation 101, lunar_latitude 109, lunar_longitude 109, mod 15, moon_node 107, mt 92, poly 34, refraction 92, right_ascension 92, sidereal_from_moment 97, sigma 31, sin_degrees 92, and solar_anomaly 103.

2.14.4 Unit tests

```
111 <timeAndAstronomyUnitTest.py 111>≡  
    # <generated code warning 1>  
    <import for testing 6>  
    from appendixCUnitTest import AppendixCTable5TestCaseBase  
    <time and astronomy unit test 94>  
    <execute tests 5>
```

Root chunk (not used in this document).
Uses AppendixCTable5TestCaseBase 184.

2.15 Persian Calendar

```

112 <persian calendar 112>≡
#####
# persian calendar algorithms #
#####
# see lines 3844-3847 in calendrica-3.0.c1
def persian_date(year, month, day):
    """Return a Persian date data structure."""
    return [year, month, day]

# see lines 3849-3852 in calendrica-3.0.c1
PERSIAN_EPOCH = fixed_from_julian(julian_date(ce(622), MARCH, 19))

# see lines 3854-3858 in calendrica-3.0.c1
TEHRAN = location(deg(mpf(35.68)),
                  deg(mpf(51.42)),
                  mt(1100),
                  hr(3 + 1/2))

# see lines 3860-3865 in calendrica-3.0.c1
def midday_in_tehran(date):
    """Return Universal time of midday on fixed date, date, in Tehran."""
    return universal_from_standard(midday(date, TEHRAN), TEHRAN)

# see lines 3867-3876 in calendrica-3.0.c1
def persian_new_year_on_or_before(date):
    """Return the fixed date of Astronomical Persian New Year on or
    before fixed date, date."""
    approx = estimate_prior_solar_longitude(SPRING, midday_in_tehran(date))
    return next(ifloor(approx) - 1,
               lambda day: (solar_longitude(midday_in_tehran(day)) <=
                           (SPRING + deg(2))))

# see lines 3880-3898 in calendrica-3.0.c1
def fixed_from_persian(p_date):
    """Return fixed date of Astronomical Persian date, p_date."""
    month = standard_month(p_date)
    day = standard_day(p_date)
    year = standard_year(p_date)
    temp = (year - 1) if (0 < year) else year
    new_year = persian_new_year_on_or_before(PERSIAN_EPOCH + 180 +
                                             ifloor(MEAN_TROPICAL_YEAR * temp))
    return ((new_year - 1) +
            ((31 * (month - 1)) if (month <= 7) else (30 * (month - 1) + 6)) +
            day)

# see lines 3898-3918 in calendrica-3.0.c1
def persian_from_fixed(date):
    """Return Astronomical Persian date (year month day)
    corresponding to fixed date, date."""
    new_year = persian_new_year_on_or_before(date)
    y = iround((new_year - PERSIAN_EPOCH) / MEAN_TROPICAL_YEAR) + 1
    year = y if (0 < y) else (y - 1)
    day_of_year = date - fixed_from_persian(persian_date(year, 1, 1)) + 1
    month = (ceiling(day_of_year / 31)
            if (day_of_year <= 186)
            else ceiling((day_of_year - 6) / 30))
    day = date - (fixed_from_persian(persian_date(year, month, 1)) - 1)
    return persian_date(year, month, day)

```

```

# see lines 3920-3932 in calendrica-3.0.cl
def is_arithmetic_persian_leap_year(p_year):
    """Return True if p_year is a leap year on the Persian calendar."""
    y = (p_year - 474) if (0 < p_year) else (p_year - 473)
    year = mod(y, 2820) + 474
    return mod((year + 38) * 31, 128) < 31

# see lines 3934-3958 in calendrica-3.0.cl
def fixed_from_arithmetic_persian(p_date):
    """Return fixed date equivalent to Persian date p_date."""
    day = standard_day(p_date)
    month = standard_month(p_date)
    p_year = standard_year(p_date)
    y = (p_year - 474) if (0 < p_year) else (p_year - 473)
    year = mod(y, 2820) + 474
    temp = (31 * (month - 1)) if (month <= 7) else ((30 * (month - 1)) + 6)

    return ((PERSIAN_EPOCH - 1)
            + (1029983 * quotient(y, 2820))
            + (365 * (year - 1))
            + quotient((31 * year) - 5, 128)
            + temp
            + day)

# see lines 3960-3986 in calendrica-3.0.cl
def arithmetic_persian_year_from_fixed(date):
    """Return Persian year corresponding to the fixed date, date."""
    d0 = date - fixed_from_arithmetic_persian(persian_date(475, 1, 1))
    n2820 = quotient(d0, 1029983)
    d1 = mod(d0, 1029983)
    y2820 = 2820 if (d1 == 1029982) else (quotient((128 * d1) + 46878, 46751))
    year = 474 + (2820 * n2820) + y2820

    return year if (0 < year) else (year - 1)

# see lines 3988-4001 in calendrica-3.0.cl
def arithmetic_persian_from_fixed(date):
    """Return the Persian date corresponding to fixed date, date."""
    year = arithmetic_persian_year_from_fixed(date)
    day_of_year = 1 + date - fixed_from_arithmetic_persian(
        persian_date(year, 1, 1))
    month = (ceiling(day_of_year / 31)
             if (day_of_year <= 186)
             else ceiling((day_of_year - 6) / 30))
    day = date - fixed_from_arithmetic_persian(persian_date(year, month, 1)) + 1
    return persian_date(year, month, day)

# see lines 4003-4015 in calendrica-3.0.cl
def naw_ruz(g_year):
    """Return the Fixed date of Persian New Year (Naw-Ruz) in Gregorian
    year g_year."""
    persian_year = g_year - gregorian_year_from_fixed(PERSIAN_EPOCH) + 1
    y = (persian_year - 1) if (persian_year <= 0) else persian_year
    return fixed_from_persian(persian_date(y, 1, 1))

```

This code is used in chunk 3.

Defines:

- arithmetic_persian_from_fixed, used in chunk 172.
- arithmetic_persian_year_from_fixed, never used.
- fixed_from_arithmetic_persian, used in chunk 172.
- fixed_from_persian, used in chunk 172.
- is_arithmetic_persian_leap_year, never used.

midday_in_tehran, never used.
 naw_ruz, never used.
 persian_date, never used.
 PERSIAN_EPOCH, never used.
 persian_from_fixed, used in chunk 172.
 persian_new_year_on_or_before, never used.
 TEHRAN, never used.
 Uses ce 59, ceiling 86 86, deg 92, estimate_prior_solar_longitude 99, fixed_from_julian 59,
 gregorian_year_from_fixed 55, hr 92, ifloor 15, iround 15, julian_date 59, location 92,
 MARCH 53, MEAN_TROPICAL_YEAR 97, midday 92, mod 15, mt 92, next 16, quotient 14 14,
 solar_longitude 97, SPRING 99, standard_day 39, standard_month 39, standard_year 39,
 and universal_from_standard 92.

113 *<persian_calendar_unit_test 113>≡*

This definition is continued in chunk 171.
 This code is used in chunks 4 and 114.

2.15.1 Unit tests

114 *<persianCalendarUnitTest.py 114>≡*
 # <generated code warning 1>
 <import for testing 6>
 from appendixCUnitTest import AppendixCTable3TestCaseBase
 <persian_calendar_unit_test 113>
 <execute tests 5>

Root chunk (not used in this document).
 Uses AppendixCTable3TestCaseBase 164.

2.16 Bahai Calendar

```
115 <bahai calendar 115>≡
#####
# bahai calendar algorithms #
#####
# see lines 4020-4023 in calendrica-3.0.cl
def bahai_date(major, cycle, year, month, day):
    """Return a Bahai date data structure."""
    return [major, cycle, year, month, day]

# see lines 4025-4027 in calendrica-3.0.cl
def bahai_major(date):
    """Return 'major' element of a Bahai date, date."""
    return date[0]

# see lines 4029-4031 in calendrica-3.0.cl
def bahai_cycle(date):
    """Return 'cycle' element of a Bahai date, date."""
    return date[1]

# see lines 4033-4035 in calendrica-3.0.cl
def bahai_year(date):
    """Return 'year' element of a Bahai date, date."""
    return date[2]

# see lines 4037-4039 in calendrica-3.0.cl
def bahai_month(date):
    """Return 'month' element of a Bahai date, date."""
    return date[3]

# see lines 4041-4043 in calendrica-3.0.cl
def bahai_day(date):
    """Return 'day' element of a Bahai date, date."""
    return date[4]

# see lines 4045-4048 in calendrica-3.0.cl
BAHAI_EPOCH = fixed_from_gregorian(gregorian_date(1844, MARCH, 21))

# see lines 4050-4053 in calendrica-3.0.cl
AYYAM_I_HA = 0

# see lines 4055-4076 in calendrica-3.0.cl
def fixed_from_bahai(b_date):
    """Return fixed date equivalent to the Bahai date, b_date."""
    major = bahai_major(b_date)
    cycle = bahai_cycle(b_date)
    year = bahai_year(b_date)
    month = bahai_month(b_date)
    day = bahai_day(b_date)
    g_year = (361 * (major - 1) +
              19 * (cycle - 1) +
              year - 1 +
              gregorian_year_from_fixed(BAHAI_EPOCH))
    if (month == AYYAM_I_HA):
        elapsed_months = 342
    elif (month == 19):
        if (is_gregorian_leap_year(g_year + 1)):
            elapsed_months = 347
        else:
            elapsed_months = 346
```

```

else:
    elapsed_months = 19 * (month - 1)

    return (fixed_from_gregorian(gregorian_date(g_year, MARCH, 20)) +
            elapsed_months +
            day)

# see lines 4078-4111 in calendrica-3.0.cl
def bahai_from_fixed(date):
    """Return Bahai date [major, cycle, year, month, day] corresponding
    to fixed date, date."""
    g_year = gregorian_year_from_fixed(date)
    start = gregorian_year_from_fixed(BAHAI_EPOCH)
    years = (g_year - start -
              (1 if (date <= fixed_from_gregorian(
                  gregorian_date(g_year, MARCH, 20))) else 0))
    major = 1 + quotient(years, 361)
    cycle = 1 + quotient(mod(years, 361), 19)
    year = 1 + mod(years, 19)
    days = date - fixed_from_bahai(bahai_date(major, cycle, year, 1, 1))

    # month
    if (date >= fixed_from_bahai(bahai_date(major, cycle, year, 19, 1))):
        month = 19
    elif (date >= fixed_from_bahai(
        bahai_date(major, cycle, year, AYYAM_I_HA, 1))):
        month = AYYAM_I_HA
    else:
        month = 1 + quotient(days, 19)

    day = date + 1 - fixed_from_bahai(bahai_date(major, cycle, year, month, 1))

    return bahai_date(major, cycle, year, month, day)

# see lines 4113-4117 in calendrica-3.0.cl
def bahai_new_year(g_year):
    """Return fixed date of Bahai New Year in Gregorian year, g_year."""
    return fixed_from_gregorian(gregorian_date(g_year, MARCH, 21))

# see lines 4119-4122 in calendrica-3.0.cl
HAIFA = location(deg(mpf(32.82)), deg(35), mt(0), hr(2))

# see lines 4124-4130 in calendrica-3.0.cl
def sunset_in_haifa(date):
    """Return universal time of sunset of evening
    before fixed date, date in Haifa."""
    return universal_from_standard(sunset(date, HAIFA), HAIFA)

# see lines 4132-4141 in calendrica-3.0.cl
def future_bahai_new_year_on_or_before(date):
    """Return fixed date of Future Bahai New Year on or
    before fixed date, date."""
    approx = estimate_prior_solar_longitude(SPRING, sunset_in_haifa(date))
    return next(iffloor(approx) - 1,
                lambda day: (solar_longitude(sunset_in_haifa(day)) <=
                             (SPRING + deg(2))))

# see lines 4143-4173 in calendrica-3.0.cl
def fixed_from_future_bahai(b_date):

```

```

    """Return fixed date of Bahai date, b_date."""
    major = bahai_major(b_date)
    cycle = bahai_cycle(b_date)
    year = bahai_year(b_date)
    month = bahai_month(b_date)
    day = bahai_day(b_date)
    years = (361 * (major - 1)) + (19 * (cycle - 1)) + year
    if (month == 19):
        return (future_bahai_new_year_on_or_before(
            BAHAI_EPOCH +
            ifloor(MEAN_TROPICAL_YEAR * (years + 1/2))) -
            20 + day)
    elif (month == AYYAM_I_HA):
        return (future_bahai_new_year_on_or_before(
            BAHAI_EPOCH +
            ifloor(MEAN_TROPICAL_YEAR * (years - 1/2))) +
            341 + day)
    else:
        return (future_bahai_new_year_on_or_before(
            BAHAI_EPOCH +
            ifloor(MEAN_TROPICAL_YEAR * (years - 1/2))) +
            (19 * (month - 1)) + day - 1)

# see lines 4175-4201 in calendrica-3.0.c1
def future_bahai_from_fixed(date):
    """Return Future Bahai date corresponding to fixed date, date."""
    new_year = future_bahai_new_year_on_or_before(date)
    years = iround((new_year - BAHAI_EPOCH) / MEAN_TROPICAL_YEAR)
    major = 1 + quotient(years, 361)
    cycle = 1 + quotient(mod(years, 361), 19)
    year = 1 + mod(years, 19)
    days = date - new_year

    if (date >= fixed_from_future_bahai(bahai_date(major, cycle, year, 19, 1))):
        month = 19
    elif (date >= fixed_from_future_bahai(
        bahai_date(major, cycle, year, AYYAM_I_HA, 1))):
        month = AYYAM_I_HA
    else:
        month = 1 + quotient(days, 19)

    day = date + 1 - fixed_from_future_bahai(
        bahai_date(major, cycle, year, month, 1))

    return bahai_date(major, cycle, year, month, day)

# see lines 4203-4213 in calendrica-3.0.c1
def feast_of_ridvan(g_year):
    """Return Fixed date of Feast of Ridvan in Gregorian year, g_year."""
    years = g_year - gregorian_year_from_fixed(BAHAI_EPOCH)
    major = 1 + quotient(years, 361)
    cycle = 1 + quotient(mod(years, 361), 19)
    year = 1 + mod(years, 19)
    return fixed_from_future_bahai(bahai_date(major, cycle, year, 2, 13))

```

This code is used in chunk 3.

Defines:

AYYAM_I_HA, never used.

bahai_cycle, never used.
 bahai_date, used in chunk 155.
 bahai_day, never used.
 BAHAI_EPOCH, never used.
 bahai_from_fixed, used in chunk 161.
 bahai_major, never used.
 bahai_month, never used.
 bahai_new_year, never used.
 bahai_year, never used.
 feast_of_ridvan, never used.
 fixed_from_bahai, used in chunk 161.
 fixed_from_future_bahai, used in chunk 161.
 future_bahai_from_fixed, used in chunk 161.
 future_bahai_new_year_on_or_before, never used.
 HAIFA, never used.
 sunset_in_haifa, never used.
 Uses deg 92, estimate_prior_solar_longitude 99, fixed_from_gregorian 55, gregorian_date 52,
 gregorian_year_from_fixed 55, hr 92, ifloor 15, iround 15, is_gregorian_leap_year 54,
 location 92, MARCH 53, MEAN_TROPICAL_YEAR 97, mod 15, mt 92, next 16, quotient 14 14,
 solar_longitude 97, SPRING 99, start 47, sunset 92, and universal_from_standard 92.

116 $\langle \text{bahai_calendar_unit_test 116} \rangle \equiv$

This definition is continued in chunk 160.
 This code is used in chunks 4 and 117.

2.16.1 Unit tests

117 $\langle \text{bahaiCalendarUnitTest.py 117} \rangle \equiv$
 # $\langle \text{generated code warning 1} \rangle$
 $\langle \text{import for testing 6} \rangle$
 from appendixCUnitTest import AppendixCTable2TestCaseBase
 $\langle \text{bahai_calendar_unit_test 116} \rangle$
 $\langle \text{execute tests 5} \rangle$

Root chunk (not used in this document).
 Uses AppendixCTable2TestCaseBase 155.

2.17 French Revolutionary Calendar

```
118 <french revolutionary calendar 118>≡
#####
# french revolutionary calendar algorithms #
#####
# see lines 4218-4220 in calendrica-3.0.cl
def french_date(year, month, day):
    """Return a French Revolutionary date data structure."""
    return [year, month, day]

# see lines 4222-4226 in calendrica-3.0.cl
#""Fixed date of start of the French Revolutionary calendar."""
FRENCH_EPOCH = fixed_from_gregorian(gregorian_date(1792, SEPTEMBER, 22))

# see lines 4228-4233 in calendrica-3.0.cl
PARIS = location(angle(48, 50, 11),
                 angle(2, 20, 15),
                 mt(27),
                 hr(1))

# see lines 4235-4241 in calendrica-3.0.cl
def midnight_in_paris(date):
    """Return Universal Time of true midnight at the end of
       fixed date, date."""
    # tricky bug: I was using midDAY!!! So French Revolutionary was failing...
    return universal_from_standard(midnight(date + 1, PARIS), PARIS)
```

This definition is continued in chunk 120.

This code is used in chunk 3.

Defines:

french_date, used in chunks 120 and 164.

FRENCH_EPOCH, used in chunk 120.

midnight_in_paris, used in chunks 119 and 120.

PARIS, used in chunk 186.

Uses angle 92, end 47, fixed_from_gregorian 55, gregorian_date 52, hr 92, location 92, midnight 92, mt 92, SEPTEMBER 53, start 47, and universal_from_standard 92.

I add a test for midnight_in_paris: it took me a lot of time to spot that the PYTHON implementation was using midday instead of midnight!

```
119 <french revolutionary calendar unit test 119>≡
def testMidnightInParis(self):
    d = fixed_from_gregorian(gregorian_date(1992, OCTOBER, 13))
    self.assertEqual(midnight_in_paris(d), d+1)
```

This definition is continued in chunks 121 and 173.

This code is used in chunks 4 and 122.

Uses fixed_from_gregorian 55, gregorian_date 52, midnight_in_paris 118, and OCTOBER 53.


```

# see lines 4243-4252 in calendrica-3.0.cl
def french_new_year_on_or_before(date):
    """Return fixed date of French Revolutionary New Year on or
       before fixed date, date."""
    approx = estimate_prior_solar_longitude(AUTUMN, midnight_in_paris(date))
    return next(ifloor(approx) - 1,
                lambda day: AUTUMN <= solar_longitude(midnight_in_paris(day)))

# see lines 4254-4267 in calendrica-3.0.cl
def fixed_from_french(f_date):
    """Return fixed date of French Revolutionary date, f_date"""
    month = standard_month(f_date)
    day = standard_day(f_date)
    year = standard_year(f_date)
    new_year = french_new_year_on_or_before(
        ifloor(FRENCH_EPOCH +
              180 +
              MEAN_TROPICAL_YEAR * (year - 1)))
    return new_year - 1 + 30 * (month - 1) + day

# see lines 4269-4278 in calendrica-3.0.cl
def french_from_fixed(date):
    """Return French Revolutionary date of fixed date, date."""
    new_year = french_new_year_on_or_before(date)
    year = iround((new_year - FRENCH_EPOCH) / MEAN_TROPICAL_YEAR) + 1
    month = quotient(date - new_year, 30) + 1
    day = mod(date - new_year, 30) + 1
    return french_date(year, month, day)

# see lines 4280-4286 in calendrica-3.0.cl
def is_arithmetic_french_leap_year(f_year):
    """Return True if year, f_year, is a leap year on the French
       Revolutionary calendar."""
    return ((mod(f_year, 4) == 0) and
            (mod(f_year, 400) not in [100, 200, 300]) and
            (mod(f_year, 4000) != 0))

# see lines 4288-4302 in calendrica-3.0.cl
def fixed_from_arithmetic_french(f_date):
    """Return fixed date of French Revolutionary date, f_date."""
    month = standard_month(f_date)
    day = standard_day(f_date)
    year = standard_year(f_date)

    return (FRENCH_EPOCH - 1
            + 365 * (year - 1)
            + quotient(year - 1, 4)
            + quotient(year - 1, 100)
            + quotient(year - 1, 400)
            + quotient(year - 1, 4000)
            + 30 * (month - 1)
            + day)

# see lines 4304-4325 in calendrica-3.0.cl
def arithmetic_french_from_fixed(date):
    """Return French Revolutionary date [year, month, day] of fixed
       date, date."""
    approx = quotient(date - FRENCH_EPOCH + 2, 1460969/4000) + 1

```

```

year    = ((approx - 1)
           if (date <
               fixed_from_arithmetic_french(french_date(approx, 1, 1)))
           else approx)
month    = 1 + quotient(date -
                        fixed_from_arithmetic_french(french_date(year, 1, 1)), 30)
day      = date - fixed_from_arithmetic_french(
                french_date(year, month, 1)) + 1
return french_date(year, month, day)

```

This code is used in chunk 3.

Defines:

```

arithmetic_french_from_fixed, used in chunk 174.
fixed_from_arithmetic_french, used in chunk 174.
fixed_from_french, used in chunk 174.
french_from_fixed, used in chunk 174.
french_new_year_on_or_before, never used.
is_arithmetic_french_leap_year, never used.

```

Uses AUTUMN 99, estimate_prior_solar_longitude 99, french_date 118, FRENCH_EPOCH 118, ifloor 15,
iround 15, MEAN_TROPICAL_YEAR 97, midnight_in_paris 118, mod 15, next 16, quotient 14 14,
solar_longitude 97, standard_day 39, standard_month 39, and standard_year 39.

121 *<french revolutionary calendar unit test 119>+≡*

This code is used in chunks 4 and 122.

2.17.1 Unit tests

122 *<frenchRevolutionaryCalendarUnitTest.py 122>≡*
<generated code warning 1>
<import for testing 6>
from appendixCUnitTest import AppendixCTable3TestCaseBase
<french revolutionary calendar unit test 119>
<execute tests 5>

Root chunk (not used in this document).

Uses AppendixCTable3TestCaseBase 164.

2.18 Chinese Calendar

```
123 <chinese_calendar 123>≡
#####
# chinese calendar algorithms #
#####
# see lines 4330-4333 in calendrica-3.0.cl
def chinese_date(cycle, year, month, leap, day):
    """Return a Chinese date data structure."""
    return [cycle, year, month, leap, day]

# see lines 4335-4337 in calendrica-3.0.cl
def chinese_cycle(date):
    """Return 'cycle' element of a Chinese date, date."""
    return date[0]

# see lines 4339-4341 in calendrica-3.0.cl
def chinese_year(date):
    """Return 'year' element of a Chinese date, date."""
    return date[1]

# see lines 4343-4345 in calendrica-3.0.cl
def chinese_month(date):
    """Return 'month' element of a Chinese date, date."""
    return date[2]

# see lines 4347-4349 in calendrica-3.0.cl
def chinese_leap(date):
    """Return 'leap' element of a Chinese date, date."""
    return date[3]

# see lines 4351-4353 in calendrica-3.0.cl
def chinese_day(date):
    """Return 'day' element of a Chinese date, date."""
    return date[4]

# see lines 4355-4363 in calendrica-3.0.cl
def chinese_location(tee):
    """Return location of Beijing; time zone varies with time, tee."""
    year = gregorian_year_from_fixed(ifloor(tee))
    if (year < 1929):
        return location(angle(39, 55, 0), angle(116, 25, 0),
                        mt(43.5), hr(1397/180))
    else:
        return location(angle(39, 55, 0), angle(116, 25, 0),
                        mt(43.5), hr(8))

# see lines 4365-4377 in calendrica-3.0.cl
def chinese_solar_longitude_on_or_after(lam, date):
    """Return moment (Beijing time) of the first date on or after
    fixed date, date, (Beijing time) when the solar longitude
    will be 'lam' degrees."""
    tee = solar_longitude_after(lam,
                                universal_from_standard(date,
                                                            chinese_location(date)))
    return standard_from_universal(tee, chinese_location(tee))

# see lines 4379-4387 in calendrica-3.0.cl
def current_major_solar_term(date):
    """Return last Chinese major solar term (zhongqi) before
```

```

    fixed date, date."""
    s = solar_longitude(universal_from_standard(date,
                                                chinese_location(date)))
    return amod(2 + quotient(int(s), deg(30)), 12)

# see lines 4389-4397 in calendrica-3.0.c1
def major_solar_term_on_or_after(date):
    """Return moment (in Beijing) of the first Chinese major
    solar term (zhongqi) on or after fixed date, date. The
    major terms begin when the sun's longitude is a
    multiple of 30 degrees."""
    s = solar_longitude(midnight_in_china(date))
    l = mod(30 * ceiling(s / 30), 360)
    return chinese_solar_longitude_on_or_after(l, date)

# see lines 4399-4407 in calendrica-3.0.c1
def current_minor_solar_term(date):
    """Return last Chinese minor solar term (jieqi) before date, date."""
    s = solar_longitude(universal_from_standard(date,
                                                chinese_location(date)))
    return amod(3 + quotient(s - deg(15), deg(30)), 12)

# see lines 4409-4422 in calendrica-3.0.c1
def minor_solar_term_on_or_after(date):
    """Return moment (in Beijing) of the first Chinese minor solar
    term (jieqi) on or after fixed date, date. The minor terms
    begin when the sun's longitude is an odd multiple of 15 degrees."""
    s = solar_longitude(midnight_in_china(date))
    l = mod(30 * ceiling((s - deg(15)) / 30) + deg(15), 360)
    return chinese_solar_longitude_on_or_after(l, date)

# see lines 4424-4433 in calendrica-3.0.c1
def chinese_new_moon_before(date):
    """Return fixed date (Beijing) of first new moon before fixed date, date."""
    tee = new_moon_before(midnight_in_china(date))
    return ifloor(standard_from_universal(tee, chinese_location(tee)))

# see lines 4435-4444 in calendrica-3.0.c1
def chinese_new_moon_on_or_after(date):
    """Return fixed date (Beijing) of first new moon on or after
    fixed date, date."""
    tee = new_moon_at_or_after(midnight_in_china(date))
    return ifloor(standard_from_universal(tee, chinese_location(tee)))

# see lines 4446-4449 in calendrica-3.0.c1
CHINESE_EPOCH = fixed_from_gregorian(gregorian_date(-2636, FEBRUARY, 15))

# see lines 4451-4457 in calendrica-3.0.c1
def is_chinese_no_major_solar_term(date):
    """Return True if Chinese lunar month starting on date, date,
    has no major solar term."""
    return (current_major_solar_term(date) ==
            current_major_solar_term(chinese_new_moon_on_or_after(date + 1)))

# see lines 4459-4463 in calendrica-3.0.c1
def midnight_in_china(date):
    """Return Universal time of (clock) midnight at start of fixed
    date, date, in China."""
    return universal_from_standard(date, chinese_location(date))

# see lines 4465-4474 in calendrica-3.0.c1

```

```

def chinese_winter_solstice_on_or_before(date):
    """Return fixed date, in the Chinese zone, of winter solstice
    on or before fixed date, date."""
    approx = estimate_prior_solar_longitude(WINTER,
                                             midnight_in_china(date + 1))

    return next(ifloor(approx) - 1,
                lambda day: WINTER < solar_longitude(
                    midnight_in_china(1 + day)))

# see lines 4476-4500 in calendrica-3.0.cl
def chinese_new_year_in_sui(date):
    """Return fixed date of Chinese New Year in sui (period from
    solstice to solstice) containing date, date."""
    s1 = chinese_winter_solstice_on_or_before(date)
    s2 = chinese_winter_solstice_on_or_before(s1 + 370)
    next_m11 = chinese_new_moon_before(1 + s2)
    m12 = chinese_new_moon_on_or_after(1 + s1)
    m13 = chinese_new_moon_on_or_after(1 + m12)
    leap_year = iround((next_m11 - m12) / MEAN_SYNODIC_MONTH) == 12

    if (leap_year and
        (is_chinese_no_major_solar_term(m12) or is_chinese_no_major_solar_term(m13))):
        return chinese_new_moon_on_or_after(1 + m13)
    else:
        return m13

# see lines 4502-4511 in calendrica-3.0.cl
def chinese_new_year_on_or_before(date):
    """Return fixed date of Chinese New Year on or before fixed date, date."""
    new_year = chinese_new_year_in_sui(date)
    if (date >= new_year):
        return new_year
    else:
        return chinese_new_year_in_sui(date - 180)

# see lines 4513-4518 in calendrica-3.0.cl
def chinese_new_year(g_year):
    """Return fixed date of Chinese New Year in Gregorian year, g_year."""
    return chinese_new_year_on_or_before(
        fixed_from_gregorian(gregorian_date(g_year, JULY, 1)))

# see lines 4520-4565 in calendrica-3.0.cl
def chinese_from_fixed(date):
    """Return Chinese date (cycle year month leap day) of fixed date, date."""
    s1 = chinese_winter_solstice_on_or_before(date)
    s2 = chinese_winter_solstice_on_or_before(s1 + 370)
    next_m11 = chinese_new_moon_before(1 + s2)
    m12 = chinese_new_moon_on_or_after(1 + s1)
    leap_year = iround((next_m11 - m12) / MEAN_SYNODIC_MONTH) == 12

    m = chinese_new_moon_before(1 + date)
    month = amod(iround((m - m12) / MEAN_SYNODIC_MONTH) -
                 (1 if (leap_year and
                        is_chinese_prior_leap_month(m12, m)) else 0),
                 12)
    leap_month = (leap_year and
                  is_chinese_no_major_solar_term(m) and
                  (not is_chinese_prior_leap_month(m12,
                                                       chinese_new_moon_before(m))))
    elapsed_years = (ifloor(mpf(1.5) -

```

```

        (month / 12) +
        ((date - CHINESE_EPOCH) / MEAN_TROPICAL_YEAR)))
cycle = 1 + quotient(elapsed_years - 1, 60)
year = amod(elapsed_years, 60)
day = 1 + (date - m)
return chinese_date(cycle, year, month, leap_month, day)

# see lines 4567-4596 in calendrica-3.0.cl
def fixed_from_chinese(c_date):
    """Return fixed date of Chinese date, c_date."""
    cycle = chinese_cycle(c_date)
    year = chinese_year(c_date)
    month = chinese_month(c_date)
    leap = chinese_leap(c_date)
    day = chinese_day(c_date)
    mid_year = ifloor(CHINESE_EPOCH +
        (((cycle - 1) * 60) + (year - 1) + 1/2) *
        MEAN_TROPICAL_YEAR))
    new_year = chinese_new_year_on_or_before(mid_year)
    p = chinese_new_moon_on_or_after(new_year + ((month - 1) * 29))
    d = chinese_from_fixed(p)
    prior_new_moon = (p if ((month == chinese_month(d)) and
        (leap == chinese_leap(d)))
        else chinese_new_moon_on_or_after(1 + p))
    return prior_new_moon + day - 1

# see lines 4598-4607 in calendrica-3.0.cl
def is_chinese_prior_leap_month(m_prime, m):
    """Return True if there is a Chinese leap month on or after lunar
    month starting on fixed day, m_prime and at or before
    lunar month starting at fixed date, m."""
    return ((m >= m_prime) and
        (is_chinese_no_major_solar_term(m) or
        is_chinese_prior_leap_month(m_prime, chinese_new_moon_before(m))))

# see lines 4609-4615 in calendrica-3.0.cl
def chinese_name(stem, branch):
    """Return BOGUS if stem/branch combination is impossible."""
    if (mod(stem, 2) == mod(branch, 2)):
        return [stem, branch]
    else:
        return BOGUS

# see lines 4617-4619 in calendrica-3.0.cl
def chinese_stem(name):
    return name[0]

# see lines 4621-4623 in calendrica-3.0.cl
def chinese_branch(name):
    return name[1]

# see lines 4625-4629 in calendrica-3.0.cl
def chinese_sexagesimal_name(n):
    """Return the n_th name of the Chinese sexagesimal cycle."""
    return chinese_name(amod(n, 10), amod(n, 12))

```

```

# see lines 4631-4644 in calendrica-3.0.cl
def chinese_name_difference(c_name1, c_name2):
    """Return the number of names from Chinese name c_name1 to the
    next occurrence of Chinese name c_name2."""
    stem1 = chinese_stem(c_name1)
    stem2 = chinese_stem(c_name2)
    branch1 = chinese_branch(c_name1)
    branch2 = chinese_branch(c_name2)
    stem_difference = stem2 - stem1
    branch_difference = branch2 - branch1
    return 1 + mod(stem_difference - 1 +
                   25 * (branch_difference - stem_difference), 60)

# see lines 4646-4649 in calendrica-3.0.cl
# see lines 214-215 in calendrica-3.0.errata.cl
def chinese_year_name(year):
    """Return sexagesimal name for Chinese year, year, of any cycle."""
    return chinese_sexagesimal_name(year)

# see lines 4651-4655 in calendrica-3.0.cl
CHINESE_MONTH_NAME_EPOCH = 57

# see lines 4657-4664 in calendrica-3.0.cl
# see lines 211-212 in calendrica-3.0.errata.cl
def chinese_month_name(month, year):
    """Return sexagesimal name for month, month, of Chinese year, year."""
    elapsed_months = (12 * (year - 1)) + (month - 1)
    return chinese_sexagesimal_name(elapsed_months - CHINESE_MONTH_NAME_EPOCH)

# see lines 4666-4669 in calendrica-3.0.cl
CHINESE_DAY_NAME_EPOCH = rd(45)

# see lines 4671-4675 in calendrica-3.0.cl
# see lines 208-209 in calendrica-3.0.errata.cl
def chinese_day_name(date):
    """Return Chinese sexagesimal name for date, date."""
    return chinese_sexagesimal_name(date - CHINESE_DAY_NAME_EPOCH)

# see lines 4677-4687 in calendrica-3.0.cl
def chinese_day_name_on_or_before(name, date):
    """Return fixed date of latest date on or before fixed date, date, that
    has Chinese name, name."""
    return (date -
            mod(date +
                chinese_name_difference(name,
                                        chinese_sexagesimal_name(CHINESE_DAY_NAME_EPOCH)),
                60))

# see lines 4689-4699 in calendrica-3.0.cl
def dragon_festival(g_year):
    """Return fixed date of the Dragon Festival occurring in Gregorian
    year g_year."""
    elapsed_years = 1 + g_year - gregorian_year_from_fixed(CHINESE_EPOCH)
    cycle = 1 + quotient(elapsed_years - 1, 60)
    year = amod(elapsed_years, 60)

```

```

    return fixed_from_chinese(chinese_date, cycle, year, 5, false, 5)

# see lines 4701-4708 in calendrica-3.0.cl
def qing_ming(g_year):
    """Return fixed date of Qingming occurring in Gregorian year, g_year."""
    return ifloor(minor_solar_term_on_or_after(
        fixed_from_gregorian(gregorian_date(g_year, MARCH, 30))))

# see lines 4710-4722 in calendrica-3.0.cl
def chinese_age(birthdate, date):
    """Return the age at fixed date, date, given Chinese birthdate, birthdate,
    according to the Chinese custom.
    Returns BOGUS if date is before birthdate."""
    today = chinese_from_fixed(date)
    if (date >= fixed_from_chinese(birthdate)):
        return (60 * (chinese_cycle(today) - chinese_cycle(birthdate)) +
                (chinese_year(today) - chinese_year(birthdate)) + 1)
    else:
        return BOGUS

# see lines 4724-4758 in calendrica-3.0.cl
def chinese_year_marriage_augury(cycle, year):
    """Return the marriage augury type of Chinese year, year in cycle, cycle.
    0 means lichun does not occur (widow or double-blind years),
    1 means it occurs once at the end (blind),
    2 means it occurs once at the start (bright), and
    3 means it occurs twice (double-bright or double-happiness)."""
    new_year = fixed_from_chinese(chinese_date(cycle, year, 1, False, 1))
    c = (cycle + 1) if (year == 60) else cycle
    y = 1 if (year == 60) else (year + 1)
    next_new_year = fixed_from_chinese(chinese_date(c, y, 1, False, 1))
    first_minor_term = current_minor_solar_term(new_year)
    next_first_minor_term = current_minor_solar_term(next_new_year)
    if ((first_minor_term == 1) and (next_first_minor_term == 12)):
        res = 0
    elif ((first_minor_term == 1) and (next_first_minor_term != 12)):
        res = 1
    elif ((first_minor_term != 1) and (next_first_minor_term == 12)):
        res = 2
    else:
        res = 3
    return res

# see lines 4760-4769 in calendrica-3.0.cl
def japanese_location(tee):
    """Return the location for Japanese calendar; varies with moment, tee."""
    year = gregorian_year_from_fixed(ifloor(tee))
    if (year < 1888):
        # Tokyo (139 deg 46 min east) local time
        loc = location(deg(mpf(35.7)), angle(139, 46, 0),
                       mt(24), hr(9 + 143/450))
    else:
        # Longitude 135 time zone
        loc = location(deg(35), deg(135), mt(0), hr(9))
    return loc

```



```

# see lines 4771-4795 in calendrica-3.0.cl
def korean_location(tee):
    """Return the location for Korean calendar; varies with moment, tee."""
    # Seoul city hall at a varying time zone.
    if (tee < fixed_from_gregorian(gregorian_date(1908, APRIL, 1))):
        #local mean time for longitude 126 deg 58 min
        z = 3809/450
    elif (tee < fixed_from_gregorian(gregorian_date(1912, JANUARY, 1))):
        z = 8.5
    elif (tee < fixed_from_gregorian(gregorian_date(1954, MARCH, 21))):
        z = 9
    elif (tee < fixed_from_gregorian(gregorian_date(1961, AUGUST, 10))):
        z = 8.5
    else:
        z = 9
    return location(angle(37, 34, 0), angle(126, 58, 0),
                    mt(0), hr(z))

# see lines 4797-4800 in calendrica-3.0.cl
def korean_year(cycle, year):
    """Return equivalent Korean year to Chinese cycle, cycle, and year, year."""
    return (60 * cycle) + year - 364

# see lines 4802-4811 in calendrica-3.0.cl
def vietnamese_location(tee):
    """Return the location for Vietnamese calendar is Hanoi;
    varies with moment, tee. Time zone has changed over the years."""
    if (tee < gregorian_new_year(1968)):
        z = 8
    else:
        z = 7
    return location(angle(21, 2, 0), angle(105, 51, 0),
                    mt(12), hr(z))

```

This code is used in chunk 3.

Defines:

```

chinese_age, never used.
chinese_branch, never used.
chinese_cycle, never used.
chinese_date, used in chunk 175.
chinese_day, never used.
chinese_day_name, used in chunk 177.
CHINESE_DAY_NAME_EPOCH, never used.
chinese_day_name_on_or_before, never used.
CHINESE_EPOCH, never used.
chinese_from_fixed, used in chunk 177.
chinese_leap, never used.
chinese_location, never used.
chinese_month, never used.
chinese_month_name, never used.
CHINESE_MONTH_NAME_EPOCH, never used.
chinese_name, never used.
chinese_name_difference, never used.
chinese_new_moon_before, never used.
chinese_new_moon_on_or_after, never used.
chinese_new_year, never used.
chinese_new_year_in_sui, never used.
chinese_new_year_on_or_before, never used.
chinese_sexagesimal_name, never used.
chinese_solar_longitude_on_or_after, never used.
chinese_stem, never used.
chinese_winter_solstice_on_or_before, never used.

```

chinese_year, never used.
 chinese_year_marriage_augury, never used.
 chinese_year_name, never used.
 current_major_solar_term, never used.
 current_minor_solar_term, never used.
 dragon_festival, never used.
 fixed_from_chinese, used in chunk 177.
 is_chinese_no_major_solar_term, never used.
 is_chinese_prior_leap_month, never used.
 japanese_location, never used.
 korean_location, never used.
 korean_year, never used.
 major_solar_term_on_or_after, used in chunk 177.
 midnight_in_china, never used.
 minor_solar_term_on_or_after, never used.
 qing_ming, never used.
 vietnamese_location, never used.
 Uses amod 15, angle 92, APRIL 53, AUGUST 53, BOGUS 13, ceiling 86 86, deg 92, degrees 92, end 47,
 estimate_prior_solar_longitude 99, FEBRUARY 53, fixed_from_gregorian 55, gregorian_date 52,
 gregorian_new_year 56, gregorian_year_from_fixed 55, hr 92, ifloor 15, iround 15, JANUARY 53,
 JULY 53, location 92, longitude 92, MARCH 53, MEAN_SYNODIC_MONTH 97, MEAN_TROPICAL_YEAR 97,
 midnight 92, mod 15, mt 92, new_moon_at_or_after 109, new_moon_before 109, next 16,
 odd 89, quotient 14 14, rd 37, solar_longitude 97, solar_longitude_after 99,
 standard_from_universal 92, start 47, universal_from_standard 92, WINTER 99, and zone 92.

124 *<chinese calendar unit test 124>*≡
 #####
 # Tests other than the ones from Appendix C #
 #####

This definition is continued in chunk 176.
 This code is used in chunks 4 and 125.

2.18.1 Unit tests

125 *<chineseCalendarUnitTest.py 125>*≡
 # *<generated code warning 1>*
<import for testing 6>
 from appendixCUnitTest import AppendixCTable4TestCaseBase
<chinese calendar unit test 124>
<execute tests 5>

Root chunk (not used in this document).
 Uses AppendixCTable4TestCaseBase 175.

2.19 Modern Hindu Calendars

```
126 <modern hindu calendars 126>≡
#####
# modern hindu calendars algorithms #
#####
# see lines 4816-4820 in calendrica-3.0.cl
def hindu_lunar_date(year, month, leap_month, day, leap_day):
    """Return a lunar Hindu date data structure."""
    return [year, month, leap_month, day, leap_day]

# see lines 4822-4824 in calendrica-3.0.cl
def hindu_lunar_month(date):
    """Return 'month' element of a lunar Hindu date, date."""
    return date[1]

# see lines 4826-4828 in calendrica-3.0.cl
def hindu_lunar_leap_month(date):
    """Return 'leap_month' element of a lunar Hindu date, date."""
    return date[2]

# see lines 4830-4832 in calendrica-3.0.cl
def hindu_lunar_day(date):
    """Return 'day' element of a lunar Hindu date, date."""
    return date[3]

# see lines 4834-4836 in calendrica-3.0.cl
def hindu_lunar_leap_day(date):
    """Return 'leap_day' element of a lunar Hindu date, date."""
    return date[4]

# see lines 4838-4840 in calendrica-3.0.cl
def hindu_lunar_year(date):
    """Return 'year' element of a lunar Hindu date, date."""
    return date[0]

# see lines 4842-4850 in calendrica-3.0.cl
def hindu_sine_table(entry):
    """Return the value for entry in the Hindu sine table.
    Entry, entry, is an angle given as a multiplier of 225'."""
    exact = 3438 * sin_degrees(entry * angle(0, 225, 0))
    error = 0.215 * signum(exact) * signum(abs(exact) - 1716)
    return iround(exact + error) / 3438

# see lines 4852-4861 in calendrica-3.0.cl
def hindu_sine(theta):
    """Return the linear interpolation for angle, theta, in Hindu table."""
    entry = theta / angle(0, 225, 0)
    fraction = mod(entry, 1)
    return ((fraction * hindu_sine_table(ceiling(entry))) +
            ((1 - fraction) * hindu_sine_table(floor(entry))))

# see lines 4863-4873 in calendrica-3.0.cl
def hindu_arcsin(amp):
    """Return the inverse of Hindu sine function of amp."""
    if (amp < 0):
```

```

        return -hindu_arcsin(-amp)
    else:
        pos = next(0, lambda k: amp <= hindu_sine_table(k))
        below = hindu_sine_table(pos - 1)
        return (angle(0, 225, 0) *
                (pos - 1 + ((amp - below) / (hindu_sine_table(pos) - below))))

# see lines 4875-4878 in calendrica-3.0.cl
HINDU_SIDEREAL_YEAR = 365 + 279457/1080000

# see lines 4880-4883 in calendrica-3.0.cl
HINDU_CREATION = HINDU_EPOCH - 1955880000 * HINDU_SIDEREAL_YEAR

# see lines 4885-4889 in calendrica-3.0.cl
def hindu_mean_position(tee, period):
    """Return the position in degrees at moment, tee, in uniform circular
    orbit of period days."""
    return deg(360) * mod((tee - HINDU_CREATION) / period, 1)

# see lines 4891-4894 in calendrica-3.0.cl
HINDU_SIDEREAL_MONTH = 27 + 4644439/14438334

# see lines 4896-4899 in calendrica-3.0.cl
HINDU_SYNODIC_MONTH = 29 + 7087771/13358334

# see lines 4901-4904 in calendrica-3.0.cl
HINDU_ANOMALISTIC_YEAR = 1577917828000/(4320000000 - 387)

# see lines 4906-4909 in calendrica-3.0.cl
HINDU_ANOMALISTIC_MONTH = mpf(1577917828)/(57753336 - 488199)

# see lines 4911-4926 in calendrica-3.0.cl
def hindu_true_position(tee, period, size, anomalistic, change):
    """Return the longitudinal position at moment, tee.
    period is the period of mean motion in days.
    size is ratio of radii of epicycle and deferent.
    anomalistic is the period of retrograde revolution about epicycle.
    change is maximum decrease in epicycle size."""
    lam = hindu_mean_position(tee, period)
    offset = hindu_sine(hindu_mean_position(tee, anomalistic))
    contraction = abs(offset) * change * size
    equation = hindu_arcsin(offset * (size - contraction))
    return mod(lam - equation, 360)

# see lines 4928-4932 in calendrica-3.0.cl
def hindu_solar_longitude(tee):
    """Return the solar longitude at moment, tee."""
    return hindu_true_position(tee,
                              HINDU_SIDEREAL_YEAR,
                              14/360,
                              HINDU_ANOMALISTIC_YEAR,
                              1/42)

# see lines 4934-4938 in calendrica-3.0.cl
def hindu_zodiac(tee):
    """Return the zodiacal sign of the sun, as integer in range 1..12,
    at moment tee."""
    return quotient(float(hindu_solar_longitude(tee)), deg(30)) + 1

```

```

# see lines 4940-4944 in calendrica-3.0.cl
def hindu_lunar_longitude(tee):
    """Return the lunar longitude at moment, tee."""
    return hindu_true_position(tee,
                                HINDU_SIDEREAL_MONTH,
                                32/360,
                                HINDU_ANOMALISTIC_MONTH,
                                1/96)

# see lines 4946-4952 in calendrica-3.0.cl
def hindu_lunar_phase(tee):
    """Return the longitudinal distance between the sun and moon
    at moment, tee."""
    return mod(hindu_lunar_longitude(tee) - hindu_solar_longitude(tee), 360)

# see lines 4954-4958 in calendrica-3.0.cl
def hindu_lunar_day_from_moment(tee):
    """Return the phase of moon (tithi) at moment, tee, as an integer in
    the range 1..30."""
    return quotient(hindu_lunar_phase(tee), deg(12)) + 1

# see lines 4960-4973 in calendrica-3.0.cl
def hindu_new_moon_before(tee):
    """Return the approximate moment of last new moon preceding moment, tee,
    close enough to determine zodiacal sign."""
    varepsilon = pow(2, -1000)
    tau = tee - ((1/deg(360)) *
                 hindu_lunar_phase(tee) *
                 HINDU_SYNODIC_MONTH)
    return binary_search(tau - 1, min(tee, tau + 1),
                        lambda l, u: ((hindu_zodiac(l) == hindu_zodiac(u)) or
                                       ((u - l) < varepsilon)),
                        lambda x: hindu_lunar_phase(x) < deg(180))

# see lines 4975-4988 in calendrica-3.0.cl
def hindu_lunar_day_at_or_after(k, tee):
    """Return the time lunar_day (tithi) number, k, begins at or after
    moment, tee. k can be fractional (for karanas)."""
    phase = (k - 1) * deg(12)
    tau = tee + ((1/deg(360)) *
                 mod(phase - hindu_lunar_phase(tee), 360) *
                 HINDU_SYNODIC_MONTH)
    a = max(tee, tau - 2)
    b = tau + 2
    return invert_angular(hindu_lunar_phase, phase, a, b)

# see lines 4990-4996 in calendrica-3.0.cl
def hindu_calendar_year(tee):
    """Return the solar year at given moment, tee."""
    return iround(((tee - HINDU_EPOCH) / HINDU_SIDEREAL_YEAR) -
                  (hindu_solar_longitude(tee) / deg(360)))

# see lines 4998-5001 in calendrica-3.0.cl

```

```

HINDU_SOLAR_ERA = 3179

# see lines 5003-5020 in calendrica-3.0.c1
def hindu_solar_from_fixed(date):
    """Return the Hindu (Orissa) solar date equivalent to fixed date, date."""
    critical = hindu_sunrise(date + 1)
    month = hindu_zodiac(critical)
    year = hindu_calendar_year(critical) - HINDU_SOLAR_ERA
    approx = date - 3 - mod(floor(hindu_solar_longitude(critical)), deg(30))
    begin = next(approx,
                  lambda i: (hindu_zodiac(hindu_sunrise(i + 1)) == month))
    day = date - begin + 1
    return hindu_solar_date(year, month, day)

# see lines 5022-5039 in calendrica-3.0.c1
def fixed_from_hindu_solar(s_date):
    """Return the fixed date corresponding to Hindu solar date, s_date,
    (Saka era; Orissa rule.)"""
    month = standard_month(s_date)
    day = standard_day(s_date)
    year = standard_year(s_date)
    begin = floor((year + HINDU_SOLAR_ERA + ((month - 1)/12)) *
                  HINDU_SIDEREAL_YEAR + HINDU_EPOCH)
    return (day - 1 +
            next(begin - 3,
                  lambda d: (hindu_zodiac(hindu_sunrise(d + 1)) == month)))

# see lines 5041-5044 in calendrica-3.0.c1
HINDU_LUNAR_ERA = 3044

# see lines 5046-5074 in calendrica-3.0.c1
def hindu_lunar_from_fixed(date):
    """Return the Hindu lunar date, new_moon scheme,
    equivalent to fixed date, date."""
    critical = hindu_sunrise(date)
    day = hindu_lunar_day_from_moment(critical)
    leap_day = (day == hindu_lunar_day_from_moment(hindu_sunrise(date - 1)))
    last_new_moon = hindu_new_moon_before(critical)
    next_new_moon = hindu_new_moon_before(floor(last_new_moon) + 35)
    solar_month = hindu_zodiac(last_new_moon)
    leap_month = (solar_month == hindu_zodiac(next_new_moon))
    month = amod(solar_month + 1, 12)
    year = (hindu_calendar_year((date + 180) if (month <= 2) else date) -
            HINDU_LUNAR_ERA)
    return hindu_lunar_date(year, month, leap_month, day, leap_day)

# see lines 5076-5123 in calendrica-3.0.c1
def fixed_from_hindu_lunar(l_date):
    """Return the Fixed date corresponding to Hindu lunar date, l_date."""
    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)
    approx = HINDU_EPOCH + (HINDU_SIDEREAL_YEAR *
                            (year + HINDU_LUNAR_ERA + ((month - 1) / 12)))
    s = floor(approx - ((1/deg(360)) *
                       HINDU_SIDEREAL_YEAR *

```

```

        mod(hindu_solar_longitude(approx) -
            ((month - 1) * deg(30)) +
            deg(180), 360) -
            deg(180)))
k = hindu_lunar_day_from_moment(s + hr(6))
if (3 < k < 27):
    temp = k
else:
    mid = hindu_lunar_from_fixed(s - 15)
    if ((hindu_lunar_month(mid) != month) or
        (hindu_lunar_leap_month(mid) and not leap_month)):
        temp = mod(k + 15, 30) - 15
    else:
        temp = mod(k - 15, 30) + 15

est = s + day - temp
tau = (est -
        mod(hindu_lunar_day_from_moment(est + hr(6)) - day + 15, 30) +
        15)
date = next(tau - 1,
            lambda d: (hindu_lunar_day_from_moment(hindu_sunrise(d)) in
                        [day, amod(day + 1, 30)]))
return (date + 1) if leap_day else date

# see lines 5125-5139 in calendrica-3.0.cl
def hindu_equation_of_time(date):
    """Return the time from true to mean midnight of date, date."""
    offset = hindu_sine(hindu_mean_position(date, HINDU_ANOMALISTIC_YEAR))
    equation_sun = (offset *
                    angle(57, 18, 0) *
                    (14/360 - (abs(offset) / 1080)))
    return ((hindu_daily_motion(date) / deg(360)) *
            (equation_sun / deg(360)) *
            HINDU_SIDEREAL_YEAR)

# see lines 5141-5155 in calendrica-3.0.cl
def hindu_ascensional_difference(date, location):
    """Return the difference between right and oblique ascension
    of sun on date, date, at loacel, location."""
    sin_delta = (1397/3438) * hindu_sine(hindu_tropical_longitude(date))
    phi = latitude(location)
    diurnal_radius = hindu_sine(deg(90) + hindu_arcsin(sin_delta))
    tan_phi = hindu_sine(phi) / hindu_sine(deg(90) + phi)
    earth_sine = sin_delta * tan_phi
    return hindu_arcsin(-earth_sine / diurnal_radius)

# see lines 5157-5172 in calendrica-3.0.cl
def hindu_tropical_longitude(date):
    """Return the Hindu tropical longitude on fixed date, date.
    Assumes precession with maximum of 27 degrees
    and period of 7200 sidereal years (= 1577917828/600 days)."""
    days = ifloor(date - HINDU_EPOCH)
    precession = (deg(27) -
                  (abs(deg(54) -
                      mod(deg(27) +
                          (deg(108) * 600/1577917828 * days),
                          108))))
    return mod(hindu_solar_longitude(date) - precession, 360)

```

```

# see lines 5174-5183 in calendrica-3.0.c1
def hindu_rising_sign(date):
    """Return the tabulated speed of rising of current zodiacal sign on
    date, date."""
    i = quotient(float(hindu_tropical_longitude(date)), deg(30))
    return [1670/1800, 1795/1800, 1935/1800, 1935/1800,
            1795/1800, 1670/1800][mod(i, 6)]

# see lines 5185-5200 in calendrica-3.0.c1
def hindu_daily_motion(date):
    """Return the sidereal daily motion of sun on date, date."""
    mean_motion = deg(360) / HINDU_SIDEREAL_YEAR
    anomaly      = hindu_mean_position(date, HINDU_ANOMALISTIC_YEAR)
    epicycle     = 14/360 - abs(hindu_sine(anomaly)) / 1080
    entry        = quotient(float(anomaly), angle(0, 225, 0))
    sine_table_step = hindu_sine_table(entry + 1) - hindu_sine_table(entry)
    factor = -3438/225 * sine_table_step * epicycle
    return mean_motion * (factor + 1)

# see lines 5202-5205 in calendrica-3.0.c1
def hindu_solar_sidereal_difference(date):
    """Return the difference between solar and sidereal day on date, date."""
    return hindu_daily_motion(date) * hindu_rising_sign(date)

# see lines 5207-5211 in calendrica-3.0.c1
UJJAIN = location(angle(23, 9, 0), angle(75, 46, 6),
                  mt(0), hr(5 + 461/9000))

# see lines 5213-5216 in calendrica-3.0.c1
# see lines 217-218 in calendrica-3.0.errata.c1
HINDU_LOCATION = UJJAIN

# see lines 5218-5228 in calendrica-3.0.c1
def hindu_sunrise(date):
    """Return the sunrise at hindu_location on date, date."""
    return (date + hr(6) +
            ((longitude(UJJAIN) - longitude(HINDU_LOCATION)) / deg(360)) -
            hindu_equation_of_time(date) +
            ((1577917828/1582237828 / deg(360)) *
            (hindu_ascensional_difference(date, HINDU_LOCATION) +
            (1/4 * hindu_solar_sidereal_difference(date)))))

# see lines 5230-5244 in calendrica-3.0.c1
def hindu_fullmoon_from_fixed(date):
    """Return the Hindu lunar date, full_moon scheme,
    equivalent to fixed date, date."""
    l_date      = hindu_lunar_from_fixed(date)
    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 = (hindu_lunar_month(hindu_lunar_from_fixed(date + 20))
        if (day >= 16)
        else month)

```



```

    return hindu_lunar_date(year, m, leap_month, day, leap_day)

# see lines 5246-5255 in calendrica-3.0.c1
def is_hindu_expunged(l_month, l_year):
    """Return True if Hindu lunar month l_month in year, l_year
    is expunged."""
    return (l_month !=
            hindu_lunar_month(
                hindu_lunar_from_fixed(
                    fixed_from_hindu_lunar(
                        [l_year, l_month, False, 15, False])))

# see lines 5257-5272 in calendrica-3.0.c1
def fixed_from_hindu_fullmoon(l_date):
    """Return the fixed date equivalent to Hindu lunar date, l_date,
    in full_moon scheme."""
    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)
    if (leap_month or (day <= 15)):
        m = month
    elif (is_hindu_expunged(amod(month - 1, 12), year)):
        m = amod(month - 2, 12)
    else:
        m = amod(month - 1, 12)
    return fixed_from_hindu_lunar(
        hindu_lunar_date(year, m, leap_month, day, leap_day))

# see lines 5274-5280 in calendrica-3.0.c1
def alt_hindu_sunrise(date):
    """Return the astronomical sunrise at Hindu location on date, date,
    per Lahiri, rounded to nearest minute, as a rational number."""
    rise = dawn(date, HINDU_LOCATION, angle(0, 47, 0))
    return 1/24 * 1/60 * iround(rise * 24 * 60)

# see lines 5282-5292 in calendrica-3.0.c1
def hindu_sunset(date):
    """Return sunset at HINDU_LOCATION on date, date."""
    return (date + hr(18) +
            ((longitude(UJJAIN) - longitude(HINDU_LOCATION)) / deg(360)) -
            hindu_equation_of_time(date) +
            (((1577917828/1582237828) / deg(360)) *
             (- hindu_ascensional_difference(date, HINDU_LOCATION) +
              (3/4 * hindu_solar_sidereal_difference(date)))))

# see lines 5294-5313 in calendrica-3.0.c1
def hindu_sundial_time(tee):
    """Return Hindu local time of temporal moment, tee."""
    date = fixed_from_moment(tee)
    time = mod(tee, 1)
    q     = ifloor(4 * time)
    if (q == 0):
        a = hindu_sunset(date - 1)
        b = hindu_sunrise(date)

```

```

        t = hr(-6)
    elif (q == 3):
        a = hindu_sunset(date)
        b = hindu_sunrise(date + 1)
        t = hr(18)
    else:
        a = hindu_sunrise(date)
        b = hindu_sunset(date)
        t = hr(6)
    return a + (2 * (b - a) * (time - t))

# see lines 5315-5318 in calendrica-3.0.c1
def ayanamsha(tee):
    """Return the difference between tropical and sidereal solar longitude."""
    return solar_longitude(tee) - sidereal_solar_longitude(tee)

# see lines 5320-5323 in calendrica-3.0.c1
def astro_hindu_sunset(date):
    """Return the geometrical sunset at Hindu location on date, date."""
    return dusk(date, HINDU_LOCATION, deg(0))

# see lines 5325-5329 in calendrica-3.0.c1
def sidereal_zodiac(tee):
    """Return the sidereal zodiacal sign of the sun, as integer in range
    1..12, at moment, tee."""
    return quotient(int(sidereal_solar_longitude(tee)), deg(30)) + 1

# see lines 5331-5337 in calendrica-3.0.c1
def astro_hindu_calendar_year(tee):
    """Return the astronomical Hindu solar year KY at given moment, tee."""
    return iround(((tee - HINDU_EPOCH) / MEAN_SIDEREAL_YEAR) -
                  (sidereal_solar_longitude(tee) / deg(360)))

# see lines 5339-5357 in calendrica-3.0.c1
def astro_hindu_solar_from_fixed(date):
    """Return the Astronomical Hindu (Tamil) solar date equivalent to
    fixed date, date."""
    critical = astro_hindu_sunset(date)
    month = sidereal_zodiac(critical)
    year = astro_hindu_calendar_year(critical) - HINDU_SOLAR_ERA
    approx = (date - 3 -
              mod(floor(sidereal_solar_longitude(critical)), deg(30)))
    begin = next(approx,
                 lambda i: (sidereal_zodiac(astro_hindu_sunset(i)) == month))
    day = date - begin + 1
    return hindu_solar_date(year, month, day)

# see lines 5359-5375 in calendrica-3.0.c1
def fixed_from_astro_hindu_solar(s_date):
    """Return the fixed date corresponding to Astronomical
    Hindu solar date (Tamil rule; Saka era)."""
    month = standard_month(s_date)
    day = standard_day(s_date)
    year = standard_year(s_date)
    approx = (HINDU_EPOCH - 3 +

```

```

        ifloor(((year + HINDU_SOLAR_ERA) + ((month - 1) / 12)) *
               MEAN_SIDEREAL_YEAR))
begin = next(approx,
             lambda i: (sidereal_zodiac(astro_hindu_sunset(i)) == month))
return begin + day - 1

# see lines 5377-5381 in calendrica-3.0.c1
def astro_lunar_day_from_moment(tee):
    """Return the phase of moon (tithi) at moment, tee, as an integer in
    the range 1..30."""
    return quotient(lunar_phase(tee), deg(12)) + 1

# see lines 5383-5410 in calendrica-3.0.c1
def astro_hindu_lunar_from_fixed(date):
    """Return the astronomical Hindu lunar date equivalent to
    fixed date, date."""
    critical = alt_hindu_sunrise(date)
    day = astro_lunar_day_from_moment(critical)
    leap_day = (day == astro_lunar_day_from_moment(
        alt_hindu_sunrise(date - 1)))
    last_new_moon = new_moon_before(critical)
    next_new_moon = new_moon_at_or_after(critical)
    solar_month = sidereal_zodiac(last_new_moon)
    leap_month = solar_month == sidereal_zodiac(next_new_moon)
    month = amod(solar_month + 1, 12)
    year = astro_hindu_calendar_year((date + 180)
                                     if (month <= 2)
                                     else date) - HINDU_LUNAR_ERA
    return hindu_lunar_date(year, month, leap_month, day, leap_day)

# see lines 5412-5460 in calendrica-3.0.c1
def fixed_from_astro_hindu_lunar(l_date):
    """Return the fixed date corresponding to Hindu lunar date, l_date."""
    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)
    approx = (HINDU_EPOCH +
              MEAN_SIDEREAL_YEAR *
              (year + HINDU_LUNAR_ERA + ((month - 1) / 12)))
    s = ifloor(approx -
               1/deg(360) * MEAN_SIDEREAL_YEAR *
               (mod(sidereal_solar_longitude(approx) -
                 (month - 1) * deg(30) + deg(180), 360) - deg(180)))
    k = astro_lunar_day_from_moment(s + hr(6))
    if (3 < k < 27):
        temp = k
    else:
        mid = astro_hindu_lunar_from_fixed(s - 15)
        if ((hindu_lunar_month(mid) != month) or
            (hindu_lunar_leap_month(mid) and not leap_month)):
            temp = mod(k + 15, 30) - 15
        else:
            temp = mod(k - 15, 30) + 15
    est = s + day - temp
    tau = (est -
           mod(astro_lunar_day_from_moment(est + hr(6)) - day + 15, 30) +

```

```

15)
date = next(tau - 1,
            lambda d: (astro_lunar_day_from_moment(alt_hindu_sunrise(d)) in
                        [day, amod(day + 1, 30)]))
return (date + 1) if leap_day else date

# see lines 5462-5467 in calendrica-3.0.cl
def hindu_lunar_station(date):
    """Return the Hindu lunar station (nakshatra) at sunrise on date, date."""
    critical = hindu_sunrise(date)
    return quotient(hindu_lunar_longitude(critical), angle(0, 800, 0)) + 1

# see lines 5469-5480 in calendrica-3.0.cl
def hindu_solar_longitude_at_or_after(lam, tee):
    """Return the moment of the first time at or after moment, tee
    when Hindu solar longitude will be lam degrees."""
    tau = tee + (HINDU_SIDEREAL_YEAR *
                (1 / deg(360)) *
                mod(lam - hindu_solar_longitude(tee), 360))
    a = max(tee, tau - 5)
    b = tau + 5
    return invert_angular(hindu_solar_longitude, lam, a, b)

# see lines 5482-5487 in calendrica-3.0.cl
def mesha_samkranti(g_year):
    """Return the fixed moment of Mesha samkranti (Vernal equinox)
    in Gregorian year, g_year."""
    jan1 = gregorian_new_year(g_year)
    return hindu_solar_longitude_at_or_after(deg(0), jan1)

# see lines 5489-5493 in calendrica-3.0.cl
SIDEREAL_START = precession(universal_from_local(mesha_samkranti(ce(285)),
                                                  HINDU_LOCATION))

# see lines 5495-5513 in calendrica-3.0.cl
def hindu_lunar_new_year(g_year):
    """Return the fixed date of Hindu lunisolar new year in
    Gregorian year, g_year."""
    jan1 = gregorian_new_year(g_year)
    mina = hindu_solar_longitude_at_or_after(deg(330), jan1)
    new_moon = hindu_lunar_day_at_or_after(1, mina)
    h_day = ifloor(new_moon)
    critical = hindu_sunrise(h_day)
    return (h_day +
            (0 if ((new_moon < critical) or
                  (hindu_lunar_day_from_moment(hindu_sunrise(h_day + 1)) == 2))
             else 1))

# see lines 5515-5539 in calendrica-3.0.cl
def is_hindu_lunar_on_or_before(l_date1, l_date2):
    """Return True if Hindu lunar date, l_date1 is on or before
    Hindu lunar date, l_date2."""
    month1 = hindu_lunar_month(l_date1)
    month2 = hindu_lunar_month(l_date2)
    leap1 = hindu_lunar_leap_month(l_date1)
    leap2 = hindu_lunar_leap_month(l_date2)

```

```

day1 = hindu_lunar_day(l_date1)
day2 = hindu_lunar_day(l_date2)
leap_day1 = hindu_lunar_leap_day(l_date1)
leap_day2 = hindu_lunar_leap_day(l_date2)
year1 = hindu_lunar_year(l_date1)
year2 = hindu_lunar_year(l_date2)
return ((year1 < year2) or
        ((year1 == year2) and
         ((month1 < month2) or
          ((month1 == month2) and
           ((leap1 and not leap2) or
            ((leap1 == leap2) and
             ((day1 < day2) or
              ((day1 == day2) and
               ((not leap_day1) or
                leap_day2))))))))))

# see lines 5941-5967 in calendrica-3.0.c1
def hindu_date_occur(l_month, l_day, l_year):
    """Return the fixed date of occurrence of Hindu lunar month, l_month,
    day, l_day, in Hindu lunar year, l_year, taking leap and
    expunged days into account. When the month is
    expunged, then the following month is used."""
    lunar = hindu_lunar_date(l_year, l_month, False, l_day, False)
    ttry = fixed_from_hindu_lunar(lunar)
    mid = hindu_lunar_from_fixed((ttry - 5) if (l_day > 15) else ttry)
    expunged = l_month != hindu_lunar_month(mid)
    l_date = hindu_lunar_date(hindu_lunar_year(mid),
                              hindu_lunar_month(mid),
                              hindu_lunar_leap_month(mid),
                              l_day,
                              False)

    if (expunged):
        return next(ttry,
                    lambda d: (not is_hindu_lunar_on_or_before(
                        hindu_lunar_from_fixed(d),
                        l_date))) - 1
    elif (l_day != hindu_lunar_day(hindu_lunar_from_fixed(ttry))):
        return ttry - 1
    else:
        return ttry

# see lines 5969-5980 in calendrica-3.0.c1
def hindu_lunar_holiday(l_month, l_day, g_year):
    """Return the list of fixed dates of occurrences of Hindu lunar
    month, month, day, day, in Gregorian year, g_year."""
    l_year = hindu_lunar_year(
        hindu_lunar_from_fixed(gregorian_new_year(g_year)))
    date1 = hindu_date_occur(l_month, l_day, l_year)
    date2 = hindu_date_occur(l_month, l_day, l_year + 1)
    return list_range([date1, date2], gregorian_year_range(g_year))

# see lines 5582-5586 in calendrica-3.0.c1
def diwali(g_year):
    """Return the list of fixed date(s) of Diwali in Gregorian year, g_year."""
    return hindu_lunar_holiday(8, 1, g_year)

```

```

# see lines 5588-5605 in calendrica-3.0.c1
def hindu_tithi_occur(l_month, tithi, tee, l_year):
    """Return the fixed date of occurrence of Hindu lunar tithi prior
    to sundial time, tee, in Hindu lunar month, l_month, and
    year, l_year."""
    approx = hindu_date_occur(l_month, ifloor(tithi), l_year)
    lunar = hindu_lunar_day_at_or_after(tithi, approx - 2)
    ttry = fixed_from_moment(lunar)
    tee_h = standard_from_sundial(ttry + tee, UJJAIN)
    if ((lunar <= tee_h) or
        (hindu_lunar_phase(standard_from_sundial(ttry + 1 + tee, UJJAIN)) >
         (12 * tithi))):
        return ttry
    else:
        return ttry + 1

# see lines 5607-5620 in calendrica-3.0.c1
def hindu_lunar_event(l_month, tithi, tee, g_year):
    """Return the list of fixed dates of occurrences of Hindu lunar tithi
    prior to sundial time, tee, in Hindu lunar month, l_month,
    in Gregorian year, g_year."""
    l_year = hindu_lunar_year(
        hindu_lunar_from_fixed(gregorian_new_year(g_year)))
    date1 = hindu_tithi_occur(l_month, tithi, tee, l_year)
    date2 = hindu_tithi_occur(l_month, tithi, tee, l_year + 1)
    return list_range([date1, date2],
                      gregorian_year_range(g_year))

# see lines 5622-5626 in calendrica-3.0.c1
def shiva(g_year):
    """Return the list of fixed date(s) of Night of Shiva in Gregorian
    year, g_year."""
    return hindu_lunar_event(11, 29, hr(24), g_year)

# see lines 5628-5632 in calendrica-3.0.c1
def rama(g_year):
    """Return the list of fixed date(s) of Rama's Birthday in Gregorian
    year, g_year."""
    return hindu_lunar_event(1, 9, hr(12), g_year)

# see lines 5634-5640 in calendrica-3.0.c1
def karana(n):
    """Return the number (0-10) of the name of the n-th (1-60) Hindu
    karana."""
    if (n == 1):
        return 0
    elif (n > 57):
        return n - 50
    else:
        return amod(n - 1, 7)

# see lines 5642-5648 in calendrica-3.0.c1
def yoga(date):
    """Return the Hindu yoga on date, date."""
    return ifloor(mod((hindu_solar_longitude(date) +
                      hindu_lunar_longitude(date)) / angle(0, 800, 0), 27)) + 1

```

```

# see lines 5650-5655 in calendrica-3.0.cl
def sacred_wednesdays(g_year):
    """Return the list of Wednesdays in Gregorian year, g_year,
    that are day 8 of Hindu lunar months."""
    return sacred_wednesdays_in_range(gregorian_year_range(g_year))

# see lines 5657-5672 in calendrica-3.0.cl
def sacred_wednesdays_in_range(range):
    """Return the list of Wednesdays within range of dates
    that are day 8 of Hindu lunar months."""
    a      = start(range)
    b      = end(range)
    wed    = kday_on_or_after(WEDNESDAY, a)
    h_date = hindu_lunar_from_fixed(wed)
    ell    = [wed] if (hindu_lunar_day(h_date) == 8) else []
    if is_in_range(wed, range):
        ell[:0] = sacred_wednesdays_in_range(interval(wed + 1, b))
        return ell
    else:
        return []

```

This code is used in chunk 3.

Defines:

```

alt_hindu_sunrise, never used.
astro_hindu_calendar_year, never used.
astro_hindu_lunar_from_fixed, used in chunk 181.
astro_hindu_solar_from_fixed, used in chunk 181.
astro_hindu_sunset, never used.
astro_lunar_day_from_moment, never used.
ayanamsha, never used.
diwali, never used.
fixed_from_astro_hindu_lunar, used in chunk 181.
fixed_from_astro_hindu_solar, used in chunk 181.
fixed_from_hindu_fullmoon, never used.
fixed_from_hindu_lunar, used in chunk 181.
fixed_from_hindu_solar, used in chunk 181.
HINDU_ANOMALISTIC_MONTH, never used.
HINDU_ANOMALISTIC_YEAR, never used.
hindu_arcsin, never used.
hindu_ascensional_difference, never used.
hindu_calendar_year, never used.
HINDU_CREATION, never used.
hindu_daily_motion, never used.
hindu_date_occur, never used.
hindu_equation_of_time, never used.
hindu_fullmoon_from_fixed, never used.
HINDU_LOCATION, never used.
hindu_lunar_date, used in chunk 175.
hindu_lunar_day, never used.
hindu_lunar_day_at_or_after, never used.
hindu_lunar_day_from_moment, never used.
HINDU_LUNAR_ERA, never used.
hindu_lunar_event, never used.
hindu_lunar_from_fixed, used in chunk 181.
hindu_lunar_holiday, never used.
hindu_lunar_leap_day, never used.
hindu_lunar_leap_month, never used.
hindu_lunar_longitude, never used.
hindu_lunar_month, never used.
hindu_lunar_new_year, never used.
hindu_lunar_phase, never used.
hindu_lunar_station, never used.
hindu_lunar_year, never used.
hindu_mean_position, never used.

```

hindu_new_moon_before, never used.
 hindu_rising_sign, never used.
 HINDU_SIDEREAL_MONTH, never used.
 HINDU_SIDEREAL_YEAR, never used.
 hindu_sine, never used.
 hindu_sine_table, never used.
 HINDU_SOLAR_ERA, never used.
 hindu_solar_from_fixed, used in chunk 181.
 hindu_solar_longitude, never used.
 hindu_solar_longitude_at_or_after, never used.
 hindu_solar_sidereal_difference, never used.
 hindu_sundial_time, never used.
 hindu_sunrise, never used.
 hindu_sunset, never used.
 HINDU_SYNODIC_MONTH, never used.
 hindu_tithi_occur, never used.
 hindu_tropical_longitude, never used.
 hindu_true_position, never used.
 hindu_zodiac, never used.
 is_hindu_expunged, never used.
 is_hindu_lunar_on_or_before, never used.
 karana, never used.
 mesha_samkranti, never used.
 rama, never used.
 sacred_wednesdays, never used.
 sacred_wednesdays_in_range, never used.
 shiva, never used.
 SIDEREAL_START, used in chunks 99 and 109.
 sidereal_zodiac, never used.
 UJJAIN, never used.
 yoga, never used.
 Uses amod 15, angle 92, binary_search 25, ce 59, ceiling 86 86, dawn 92, deg 92, degrees 92, dusk 92, end 47, fixed_from_moment 40, gregorian_new_year 56, gregorian_year_range 56, HINDU_EPOCH 86, hindu_solar_date 86, hr 92, ifloor 15, interval 47, invert_angular 28 28, iround 15, is_in_range 47, kday_on_or_after 56, latitude 92, list_range 47, location 92, longitude 92, lunar_phase 109, MEAN_SIDEREAL_YEAR 97, midnight 92, minute 39, mod 15, mt 92, new_moon_at_or_after 109, new_moon_before 109, next 16, precession 99, quotient 14 14, sidereal_solar_longitude 99, signum 92, sin_degrees 92, solar_longitude 97, standard_day 39, standard_from_sundial 95, standard_month 39, standard_year 39, start 47, sunrise 92, sunset 92, universal_from_local 92, and WEDNESDAY 38.

127 *<modern hindu calendars unit test 127>≡*

This definition is continued in chunk 180.
 This code is used in chunks 4 and 128.

2.19.1 Unit tests

128 *<modernHinduCalendarsUnitTest.py 128>≡*
 # *<generated code warning 1>*
<import for testing 6>
 from appendixCUnitTest import AppendixCTable4TestCaseBase
<modern hindu calendars unit test 127>
<execute tests 5>

Root chunk (not used in this document).
 Uses AppendixCTable4TestCaseBase 175.

2.20 Tibetan Calendar

```
129 <tibetan calendar 129>≡
#####
# tibetan calendar algorithms #
#####
# see lines 5677-5681 in calendrica-3.0.cl
def tibetan_date(year, month, leap_month, day, leap_day):
    """Return a Tibetan date data structure."""
    return [year, month, leap_month, day, leap_day]

# see lines 5683-5685 in calendrica-3.0.cl
def tibetan_month(date):
    """Return 'month' element of a Tibetan date, date."""
    return date[1]

# see lines 5687-5689 in calendrica-3.0.cl
def tibetan_leap_month(date):
    """Return 'leap month' element of a Tibetan date, date."""
    return date[2]

# see lines 5691-5693 in calendrica-3.0.cl
def tibetan_day(date):
    """Return 'day' element of a Tibetan date, date."""
    return date[3]

# see lines 5695-5697 in calendrica-3.0.cl
def tibetan_leap_day(date):
    """Return 'leap day' element of a Tibetan date, date."""
    return date[4]

# see lines 5699-5701 in calendrica-3.0.cl
def tibetan_year(date):
    """Return 'year' element of a Tibetan date, date."""
    return date[0]

# see lines 5703-5705 in calendrica-3.0.cl
TIBETAN_EPOCH = fixed_from_gregorian(gregorian_date(-127, DECEMBER, 7))

# see lines 5707-5717 in calendrica-3.0.cl
def tibetan_sun_equation(alpha):
    """Return the interpolated tabular sine of solar anomaly, alpha."""
    if (alpha > 6):
        return -tibetan_sun_equation(alpha - 6)
    elif (alpha > 3):
        return tibetan_sun_equation(6 - alpha)
    elif isinstance(alpha, int):
        return [0, 6/60, 10/60, 11/60][alpha]
    else:
        return ((mod(alpha, 1) * tibetan_sun_equation(ceiling(alpha))) +
                (mod(-alpha, 1) * tibetan_sun_equation(ifloor(alpha))))

# see lines 5719-5731 in calendrica-3.0.cl
def tibetan_moon_equation(alpha):
    """Return the interpolated tabular sine of lunar anomaly, alpha."""
    if (alpha > 14):
        return -tibetan_moon_equation(alpha - 14)
    elif (alpha > 7):
```

```

        return tibetan_moon_equation(14 -alpha)
    elif isinstance(alpha, int):
        return [0, 5/60, 10/60, 15/60,
                19/60, 22/60, 24/60, 25/60][alpha]
    else:
        return ((mod(alpha, 1) * tibetan_moon_equation(ceiling(alpha))) +
                (mod(-alpha, 1) * tibetan_moon_equation(ifloor(alpha))))

# see lines 5733-5755 in calendrica-3.0.c1
def fixed_from_tibetan(t_date):
    """Return the fixed date corresponding to Tibetan lunar date, t_date."""
    year      = tibetan_year(t_date)
    month     = tibetan_month(t_date)
    leap_month = tibetan_leap_month(t_date)
    day       = tibetan_day(t_date)
    leap_day  = tibetan_leap_day(t_date)
    months = ifloor((804/65 * (year - 1)) +
                    (67/65 * month) +
                    (-1 if leap_month else 0) +
                    64/65)
    days = (30 * months) + day
    mean = ((days * 11135/11312) -30 +
            (0 if leap_day else -1) +
            1071/1616)
    solar_anomaly = mod((days * 13/4824) + 2117/4824, 1)
    lunar_anomaly = mod((days * 3781/105840) +
                        2837/15120, 1)
    sun  = -tibetan_sun_equation(12 * solar_anomaly)
    moon = tibetan_moon_equation(28 * lunar_anomaly)
    return ifloor(TIBETAN_EPOCH + mean + sun + moon)

# see lines 5757-5796 in calendrica-3.0.c1
def tibetan_from_fixed(date):
    """Return the Tibetan lunar date corresponding to fixed date, date."""
    cap_Y = 365 + 4975/18382
    years = ceiling((date - TIBETAN_EPOCH) / cap_Y)
    year0 = final(years,
                  lambda y: (date >=
                             fixed_from_tibetan(
                                 tibetan_date(y, 1, False, 1, False))))
    month0 = final(1,
                  lambda m: (date >=
                             fixed_from_tibetan(
                                 tibetan_date(year0, m, False, 1, False))))
    est = date - fixed_from_tibetan(
        tibetan_date(year0, month0, False, 1, False))
    day0 = final(est -2,
                  lambda d: (date >=
                             fixed_from_tibetan(
                                 tibetan_date(year0, month0, False, d, False))))
    leap_month = (day0 > 30)
    day = amod(day0, 30)
    if (day > day0):
        temp = month0 - 1
    elif leap_month:
        temp = month0 + 1
    else:
        temp = month0
    month = amod(temp, 12)

```

```

    if ((day > day0) and (month0 == 1)):
        year = year0 - 1
    elif (leap_month and (month0 == 12)):
        year = year0 + 1
    else:
        year = year0
    leap_day = date == fixed_from_tibetan(
        tibetan_date(year, month, leap_month, day, True))
    return tibetan_date(year, month, leap_month, day, leap_day)

# see lines 5798-5805 in calendrica-3.0.c1
def is_tibetan_leap_month(t_month, t_year):
    """Return True if t_month is leap in Tibetan year, t_year."""
    return (t_month ==
            tibetan_month(tibetan_from_fixed(
                fixed_from_tibetan(
                    tibetan_date(t_year, t_month, True, 2, False)))))

# see lines 5807-5813 in calendrica-3.0.c1
def losar(t_year):
    """Return the fixed date of Tibetan New Year (Losar)
    in Tibetan year, t_year."""
    t_leap = is_tibetan_leap_month(1, t_year)
    return fixed_from_tibetan(tibetan_date(t_year, 1, t_leap, 1, False))

# see lines 5815-5824 in calendrica-3.0.c1
def tibetan_new_year(g_year):
    """Return the list of fixed dates of Tibetan New Year in
    Gregorian year, g_year."""
    dec31 = gregorian_year_end(g_year)
    t_year = tibetan_year(tibetan_from_fixed(dec31))
    return list_range([losar(t_year - 1), losar(t_year)],
                      gregorian_year_range(g_year))

```

This code is used in chunk 3.

Defines:

fixed_from_tibetan, used in chunk 183.
 is_tibetan_leap_month, never used.
 losar, never used.
 tibetan_date, used in chunk 175.
 tibetan_day, never used.
 tibetan_from_fixed, used in chunk 183.
 tibetan_leap_day, never used.
 tibetan_leap_month, never used.
 tibetan_month, never used.
 tibetan_moon_equation, never used.
 tibetan_new_year, never used.
 tibetan_sun_equation, never used.
 tibetan_year, never used.

Uses amod 15, ceiling 86 86, DECEMBER 53, final 19, fixed_from_gregorian 55, gregorian_date 52,
 gregorian_year_end 56, gregorian_year_range 56, ifloor 15, list_range 47, lunar_anomaly 105,
 mod 15, and solar_anomaly 103.

130 *<tibetan calendar unit test 130>*≡

This definition is continued in chunk 182.

This code is used in chunks 4 and 131.

2.20.1 Unit tests

```
131 <tibetanCalendarUnitTest.py 131>≡  
    # <generated code warning 1>  
    <import for testing 6>  
    from appendixCTest import AppendixCTestCaseBase  
    <tibetan calendar unit test 130>  
    <execute tests 5>
```

Root chunk (not used in this document).
Uses AppendixCTestCaseBase 175.

2.21 Astronomical Lunar Calendars

```

132 <astronomical lunar calendars 95>+≡
#####
# astronomical lunar calendars algorithms #
#####
# see lines 5829-5845 in calendrica-3.0.cl
def visible_crescent(date, location):
    """Return S. K. Shaukat's criterion for likely
    visibility of crescent moon on eve of date 'date',
    at location 'location'."""
    tee = universal_from_standard(dusk(date - 1, location, deg(mpf(4.5))),
                                location)

    phase = lunar_phase(tee)
    altitude = lunar_altitude(tee, location)
    arc_of_light = arccos_degrees(cosine_degrees(lunar_latitude(tee)) *
                                cosine_degrees(phase))
    return ((NEW < phase < FIRST_QUARTER) and
            (deg(mpf(10.6)) <= arc_of_light <= deg(90)) and
            (altitude > deg(mpf(4.1))))

# see lines 5847-5860 in calendrica-3.0.cl
def phasis_on_or_before(date, location):
    """Return the closest fixed date on or before date 'date', when crescent
    moon first became visible at location 'location'."""
    mean = date - ifloor(lunar_phase(date + 1) / deg(360) *
                        MEAN_SYNODIC_MONTH)

    tau = ((mean - 30)
            if (((date - mean) <= 3) and (not visible_crescent(date, location)))
            else (mean - 2))
    return next(tau, lambda d: visible_crescent(d, location))

# see lines 5862-5866 in calendrica-3.0.cl
# see lines 220-221 in calendrica-3.0.errata.cl
# Sample location for Observational Islamic calendar
# (Cairo, Egypt).
ISLAMIC_LOCATION = location(deg(mpf(30.1)), deg(mpf(31.3)), mt(200), hr(2))

# see lines 5868-5882 in calendrica-3.0.cl
def fixed_from_observational_islamic(i_date):
    """Return fixed date equivalent to Observational Islamic date, i_date."""
    month = standard_month(i_date)
    day = standard_day(i_date)
    year = standard_year(i_date)
    midmonth = ISLAMIC_EPOCH + ifloor((((year - 1) * 12) + month - 0.5) *
                                    MEAN_SYNODIC_MONTH)
    return (phasis_on_or_before(midmonth, ISLAMIC_LOCATION) +
            day - 1)

# see lines 5884-5896 in calendrica-3.0.cl
def observational_islamic_from_fixed(date):
    """Return Observational Islamic date (year month day)
    corresponding to fixed date, date."""
    crescent = phasis_on_or_before(date, ISLAMIC_LOCATION)
    elapsed_months = iround((crescent - ISLAMIC_EPOCH) / MEAN_SYNODIC_MONTH)
    year = quotient(elapsed_months, 12) + 1
    month = mod(elapsed_months, 12) + 1
    day = (date - crescent) + 1
    return islamic_date(year, month, day)

# see lines 5898-5901 in calendrica-3.0.cl

```

```

JERUSALEM = location(deg(mpf(31.8)), deg(mpf(35.2)), mt(800), hr(2))

# see lines 5903-5918 in calendrica-3.0.c1
def astronomical_easter(g_year):
    """Return date of (proposed) astronomical Easter in Gregorian
    year, g_year."""
    jan1 = gregorian_new_year(g_year)
    equinox = solar_longitude_after(SPRING, jan1)
    paschal_moon = ifloor(apparent_from_local(
        local_from_universal(
            lunar_phase_at_or_after(FULL, equinox),
            JERUSALEM),
            JERUSALEM))
    # Return the Sunday following the Paschal moon.
    return kday_after(SUNDAY, paschal_moon)

# see lines 5920-5923 in calendrica-3.0.c1
JAFFA = location(angle(32, 1, 60), angle(34, 45, 0), mt(0), hr(2))

# see lines 5925-5938 in calendrica-3.0.c1
def phasis_on_or_after(date, location):
    """Return closest fixed date on or after date, date, on the eve
    of which crescent moon first became visible at location, location."""
    mean = date - ifloor(lunar_phase(date + 1) / deg(mpf(360))) *
        MEAN_SYNODIC_MONTH
    tau = (date if (((date - mean) <= 3) and
        (not visible_crescent(date - 1, location)))
        else (mean + 29))
    return next(tau, lambda d: visible_crescent(d, location))

# see lines 5940-5955 in calendrica-3.0.c1
def observational_hebrew_new_year(g_year):
    """Return fixed date of Observational (classical)
    Nisan 1 occurring in Gregorian year, g_year."""
    jan1 = gregorian_new_year(g_year)
    equinox = solar_longitude_after(SPRING, jan1)
    sset = universal_from_standard(sunset(ifloor(equinox), JAFFA), JAFFA)
    return phasis_on_or_after(ifloor(equinox) - (14 if (equinox < sset) else 13),
        JAFFA)

# see lines 5957-5973 in calendrica-3.0.c1
def fixed_from_observational_hebrew(h_date):
    """Return fixed date equivalent to Observational Hebrew date."""
    month = standard_month(h_date)
    day = standard_day(h_date)
    year = standard_year(h_date)
    year1 = (year - 1) if (month >= TISHRI) else year
    start = fixed_from_hebrew(hebrew_date(year1, NISAN, 1))
    g_year = gregorian_year_from_fixed(start + 60)
    new_year = observational_hebrew_new_year(g_year)
    midmonth = new_year + iround(29.5 * (month - 1)) + 15
    return phasis_on_or_before(midmonth, JAFFA) + day - 1

# see lines 5975-5991 in calendrica-3.0.c1
def observational_hebrew_from_fixed(date):
    """Return Observational Hebrew date (year month day)
    corresponding to fixed date, date."""
    crescent = phasis_on_or_before(date, JAFFA)
    g_year = gregorian_year_from_fixed(date)
    ny = observational_hebrew_new_year(g_year)
    new_year = observational_hebrew_new_year(g_year - 1) if (date < ny) else ny

```

```

month = iround((crescent - new_year) / 29.5) + 1
year = (standard_year(hebrew_from_fixed(new_year)) +
        (1 if (month >= TISHRI) else 0))
day = date - crescent + 1
return hebrew_date(year, month, day)

# see lines 5993-5997 in calendrica-3.0.cl
def classical_passover_eve(g_year):
    """Return fixed date of Classical (observational) Passover Eve
    (Nisan 14) occurring in Gregorian year, g_year."""
    return observational_hebrew_new_year(g_year) + 13

```

This code is used in chunk 3.

Defines:

astronomical_easter, used in chunk 168.
 classical_passover_eve, never used.
 fixed_from_observational_hebrew, used in chunks 166 and 243.
 fixed_from_observational_islamic, used in chunk 159.
 ISLAMIC_LOCATION, never used.
 JAFFA, used in chunk 94.
 JERUSALEM, used in chunk 186.
 observational_hebrew_from_fixed, used in chunks 166 and 243.
 observational_hebrew_new_year, never used.
 observational_islamic_from_fixed, used in chunk 159.
 phasis_on_or_after, never used.
 phasis_on_or_before, never used.
 visible_crescent, never used.

Uses angle 92, apparent_from_local 92, arccos_degrees 92, cosine_degrees 92,
 deg 92, dusk 92, FIRST_QUARTER 109, fixed_from_hebrew 80, FULL 109, gregorian_new_year 56,
 gregorian_year_from_fixed 55, hebrew_date 80, hebrew_from_fixed 80, hr 92, ifloor 15, iround 15,
 islamic_date 77, ISLAMIC_EPOCH 77, kday_after 56, local_from_universal 92, location 92,
 lunar_altitude 110, lunar_latitude 109, lunar_phase 109, lunar_phase_at_or_after 109,
 MEAN_SYNODIC_MONTH 97, mod 15, mt 92, NEW 109, next 16, NISAN 80, quotient 14 14,
 solar_longitude_after 99, SPRING 99, standard_day 39, standard_month 39, standard_year 39,
 start 47, SUNDAY 38, sunset 92, TISHRI 80, and universal_from_standard 92.

```

133  <coda 133>≡
      # That's all folks!

```

This code is used in chunk 3.

2.21.1 Unit tests

```

134  <astronomicalLunarCalendarsUnitTest.py 134>≡
      # <generated code warning 1>
      <import for testing 6>
      from appendixCUnitTest import AppendixCTable5TestCaseBase
      <astronomical lunar calendars unit test 135>
      <execute tests 5>

```

Root chunk (not used in this document).

Uses AppendixCTable5TestCaseBase 184.

```

135  <astronomical lunar calendars unit test 135>≡
      class AstronomicalLunarCalendarsTestCase(unittest.TestCase):
          <astronomical lunar tests 96>

```

This code is used in chunks 4 and 134.

Defines:

AstronomicalLunarCalendarsTestCase, never used.

2.22 Appendix C test data and unit tests

Prof. Reingold kindly provided me with an electronic version of Appendix C test data [1, see pag xxx]. In order to verify the alignment of my implementation in PYTHON with the one of the book, I designed the following tools and tests:

```
136 <transformLatexDates2Cvs 136>≡
    #/bin/sh
    # transform "Calendrical Calculations" Appendix C test data
    # from LaTeX table to comma-separated-values

    sed -e 's/\$/g' -e 's/\\//g' \
        -e 's/[ ][ ]*\\&[ ][ ]*/,/g' \
        -e 's/multicolumn{4}{c}{fun{bogus}}/bogus,bogus,bogus,bogus/g'
```

Root chunk (not used in this document).

```
137 <appendixCUnitTest.py 137>≡
    # <generated code warning 1>
    <import for testing 6>
    <appendix c unit test 138>
    <execute tests 5>
```

Root chunk (not used in this document).

```
138 <appendix c unit test 138>≡
#####
# The idea is to use the Appendix C in "Calendrical Calculations", 3rd Ed      #
# values to check the correctness (or the same erroneusness !-) of the        #
# Python implementation.                                                         #
#####

# read each row and transform the first cell in the various calendar and the
# check the result
import csv
<appendixCTable1Tests 139>
<appendixCTable2Tests 155>
<appendixCTable3Tests 164>
<appendixCTable4Tests 175>
<appendixCTable5Tests 184>
```

This code is used in chunk 137.

The test data are for five areas. The first one is about weekdays, Julian day, modified Julian day, Gregorian date, ISO date, Julian date, Roman name of Julian date, Egyptian date, Armenian date and Coptic date.

```

139 <appendixCTable1Tests 139>=
    class AppendixCTable1TestCaseBase():
        """This class provides methods to load the relevant test data and helpers."""

        def _dayOfWeek(self, d):
            return self._wdDict[d]

        def _romanLeap(self, c):
            return False if c == 'f' else True

        def setUp(self):
            self._wdDict = {'Sunday': SUNDAY,
                            'Monday': MONDAY,
                            'Tuesday': TUESDAY,
                            'Wednesday': WEDNESDAY,
                            'Thursday': THURSDAY,
                            'Friday': FRIDAY,
                            'Saturday': SATURDAY}

            reader = csv.reader(open("dates1.csv", "rU"),
                                delimiter=',',
                                quoting=csv.QUOTE_NONE)

            self.rd = [] # Rata Die
            self.wd = [] # WeekDay
            self.jd = [] # Julian Day
            self.mjd = [] # Modified Julian Day
            self.gd = [] # Gregorian Date
            self.isod = [] # ISO Date
            self.jdt = [] # Julian Date
            self.jrn = [] # Julian Roman Name
            self.ed = [] # Egyptian Date
            self.ad = [] # Armenian Date
            self.cd = [] # Coptic Date

            for row in reader:
                self.rd.append(int(row[0]))
                self.wd.append(self._dayOfWeek(row[1]))
                self.jd.append(float(row[2]))
                self.mjd.append(int(row[3]))
                self.gd.append(
                    gregorian_date(int(row[4]), int(row[5]), int(row[6])))
                self.isod.append(
                    iso_date(int(row[7]), int(row[8]), int(row[9])))
                self.jdt.append(
                    julian_date(int(row[10]), int(row[11]), int(row[12])))
                self.jrn.append(roman_date(int(row[13]),
                                            int(row[14]),
                                            int(row[15]),
                                            int(row[16]),
                                            self._romanLeap(row[17])))

                self.ed.append(
                    egyptian_date(int(row[18]), int(row[19]), int(row[20])))
                self.ad.append(
                    armenian_date(int(row[21]), int(row[22]), int(row[23])))
                self.cd.append(
                    coptic_date(int(row[24]), int(row[25]), int(row[26])))

```

```

class AppendixCTable1TestCase(AppendixCTable1TestCaseBase, unittest.TestCase):
    <test basic appendix c 141>
    <test julian appendix c 148>
    <test gregorian appendix c 145>
    <test iso appendix c 147>
    <test julian appendix c 148>
    <test egyptian appendix c 150>
    <test armenian appendix c 151>
    <test coptic appendix c 154>

```

This code is used in chunk 138.

Defines:

AppendixCTable1TestCase, never used.

AppendixCTable1TestCaseBase, used in chunks 4, 49, 58, 64, 67, 70, 73, 140, 142, 144, 146, 149, 152, and 153.

Uses **armenian_date** 65, **coptic_date** 71, **egyptian_date** 65, **FRIDAY** 38, **gregorian_date** 52, **iso_date** 68, **julian_date** 59, **MONDAY** 38, **rd** 37, **roman_date** 61, **SATURDAY** 38, **SUNDAY** 38, **THURSDAY** 38, **TUESDAY** 38, and **WEDNESDAY** 38.

```

140 <basic code unit test 50>+≡
    class BasicAppendixCTestCase(AppendixCTable1TestCaseBase, unittest.TestCase):
        <test basic appendix c 141>

```

This code is used in chunks 4 and 49.

Defines:

BasicAppendixCTestCase, never used.

Uses **AppendixCTable1TestCaseBase** 139.

```

141 <test basic appendix c 141>≡
    def testWeekdays(self):
        for i in range(len(self.rd)):
            # weekdays
            self.assertEqual(day_of_week_from_fixed(self.rd[i]), self.wd[i])

```

This code is used in chunks 139 and 140.

Defines:

testWeekdays, never used.

Uses **day_of_week_from_fixed** 38 and **rd** 37.

```

142 <julian calendar unit test 60>+≡
    class JulianDayAppendixCTestCase(AppendixCTable1TestCaseBase,
                                      unittest.TestCase):
        <test julian day appendix c 143>

```

This code is used in chunks 4 and 64.

Defines:

JulianDayAppendixCTestCase, never used.

Uses **AppendixCTable1TestCaseBase** 139.

```

143 <test julian day appendix c 143>≡
    def testJulianDay(self):
        for i in range(len(self.rd)):
            # julian day
            self.assertEqual(jd_from_fixed(self.rd[i]), self.jd[i])
            self.assertEqual(fixed_from_jd(self.jd[i]), self.rd[i])
            # modified julian day
            self.assertEqual(mjd_from_fixed(self.rd[i]), self.mjd[i])
            self.assertEqual(fixed_from_mjd(self.mjd[i]), self.rd[i])

```

This code is used in chunk 142.

Defines:

testJulianDay, never used.

Uses **fixed_from_jd** 48, **fixed_from_mjd** 48, **jd_from_fixed** 48, **mjd_from_fixed** 48, and **rd** 37.

```

144  <gregorian calendar unit test 57>+≡
      class GregorianAppendixCTestCase(AppendixCTable1TestCaseBase,
                                         unittest.TestCase):
          <test gregorian appendix c 145>

```

This code is used in chunks 4 and 58.

Defines:

GregorianAppendixCTestCase, never used.

Uses AppendixCTable1TestCaseBase 139.

```

145  <test gregorian appendix c 145>≡
      def testGregorian(self):
          for i in range(len(self.rd)):
              self.assertEqual(gregorian_from_fixed(self.rd[i]), self.gd[i])
              self.assertEqual(fixed_from_gregorian(self.gd[i]), self.rd[i])
              self.assertEqual(
                  gregorian_year_from_fixed(self.rd[i]), standard_year(self.gd[i]))

      def testAltGregorian(self):
          for i in range(len(self.rd)):
              self.assertEqual(alt_gregorian_from_fixed(self.rd[i]), self.gd[i])
              self.assertEqual(alt_fixed_from_gregorian(self.gd[i]), self.rd[i])
              self.assertEqual(
                  alt_gregorian_year_from_fixed(self.rd[i]),
                  standard_year(self.gd[i]))

```

This code is used in chunks 139 and 144.

Defines:

testAltGregorian, never used.

testGregorian, never used.

Uses alt_fixed_from_gregorian 56, alt_gregorian_from_fixed 56, alt_gregorian_year_from_fixed 56, fixed_from_gregorian 55, gregorian_from_fixed 56, gregorian_year_from_fixed 55, rd 37, and standard_year 39.

```

146  <iso calendar unit test 69>+≡
      class IsoAppendixCTestCase(AppendixCTable1TestCaseBase, unittest.TestCase):
          <test iso appendix c 147>

```

This code is used in chunks 4 and 70.

Defines:

IsoAppendixCTestCase, never used.

Uses AppendixCTable1TestCaseBase 139.

```

147  <test iso appendix c 147>≡
      def testIso(self):
          for i in range(len(self.rd)):
              self.assertEqual(iso_from_fixed(self.rd[i]), self.isod[i])
              self.assertEqual(fixed_from_iso(self.isod[i]), self.rd[i])

```

This code is used in chunks 139 and 146.

Defines:

testIso, never used.

Uses fixed_from_iso 68, iso_from_fixed 68, and rd 37.

```

148  <test julian appendix c 148>≡
      def testJulian(self):
          for i in range(len(self.rd)):
              # julian date
              self.assertEqual(julian_from_fixed(self.rd[i]), self.jdt[i])
              self.assertEqual(fixed_from_julian(self.jdt[i]), self.rd[i])
              # julian date, roman name
              self.assertEqual(roman_from_fixed(self.rd[i]), self.jrn[i])
              self.assertEqual(fixed_from_roman(self.jrn[i]), self.rd[i])

```

This code is used in chunks 139 and 149.

Defines:

testJulian, never used.

Uses fixed_from_julian 59, fixed_from_roman 61, julian_from_fixed 59, rd 37,
and roman_from_fixed 61.

```

149  <julian calendar unit test 60>+≡
      class JulianAppendixCTestCase(AppendixCTestCaseBase):
          <test julian appendix c 148>

```

This code is used in chunks 4 and 64.

Defines:

JulianAppendixCTestCase, never used.

Uses AppendixCTestCaseBase 139.

```

150  <test egyptian appendix c 150>≡
      def testEgyptian(self):
          for i in range(len(self.rd)):
              self.assertEqual(egyptian_from_fixed(self.rd[i]), self.ed[i])
              self.assertEqual(fixed_from_egyptian(self.ed[i]), self.rd[i])

```

This code is used in chunks 139 and 152.

Defines:

testEgyptian, never used.

Uses egyptian_from_fixed 65, fixed_from_egyptian 65, and rd 37.

```

151  <test armenian appendix c 151>≡
      def testArmenian(self):
          for i in range(len(self.rd)):
              self.assertEqual(armenian_from_fixed(self.rd[i]), self.ad[i])
              self.assertEqual(fixed_from_armenian(self.ad[i]), self.rd[i])

```

This code is used in chunks 139 and 152.

Defines:

testArmenian, never used.

Uses armenian_from_fixed 65, fixed_from_armenian 65, and rd 37.

```

152  <egyptian and armenian calendars unit test 66>+≡
      class ArmeniamAppendixCTestCase(AppendixCTestCaseBase):
          <test armenian appendix c 151>

      class EgyptianAppendixCTestCase(AppendixCTestCaseBase):
          <test egyptian appendix c 150>

```

This code is used in chunks 4 and 67.

Defines:

ArmeniamAppendixCTestCase, never used.

EgyptianAppendixCTestCase, never used.

Uses AppendixCTestCaseBase 139.

153 \langle coptic and ethiopic calendars unit test 72 $\rangle + \equiv$
 class CopticAppendixCTestCase(AppendixCTable1TestCaseBase):
 (test coptic appendix c 154)

This code is used in chunks 4 and 73.

Defines:

 CopticAppendixCTestCase, never used.

Uses AppendixCTable1TestCaseBase 139.

154 \langle test coptic appendix c 154 $\rangle \equiv$
 def testCoptic(self):
 for i in range(len(self.rd)):
 self.assertEqual(coptic_from_fixed(self.rd[i]), self.cd[i])
 self.assertEqual(fixed_from_coptic(self.cd[i]), self.rd[i])

This code is used in chunks 139 and 153.

Defines:

 testCoptic, never used.

Uses coptic_from_fixed 71, fixed_from_coptic 71, and rd 37.

The second is for Ethiopic, Islamic (Arithmetic and Observational), Bahai, Future Bahai, Mayan (Long count, Haab and Tzolkin), Aztec (Xihuitl and Tonalpohualli).

```

155 <appendixCTable2Tests 155>≡
    class AppendixCTable2TestCaseBase():
        def setUp(self):
            reader = csv.reader(open("dates2.csv", "rU"),
                                delimiter=',',
                                quoting=csv.QUOTE_NONE)

            self.rd = [] # Rata Die
            self.ed = [] # Ethiopic Date
            self.id = [] # Islamic Date
            self.io = [] # Islamic Observational date
            self.bd = [] # Bahai Date
            self.bf = [] # Bahai Future date
            self.mlc = [] # Mayan Long Count
            self.mh = [] # Mayan Haab
            self.mt = [] # Mayan Tzolkin
            self.ax = [] # Aztec Xihuitl
            self.at = [] # Aztec Tonalpohualli

            for row in reader:
                self.rd.append(int(row[0]))
                self.ed.append(
                    ethiopic_date(int(row[1]), int(row[2]), int(row[3])))
                self.id.append(
                    islamic_date(int(row[4]), int(row[5]), int(row[6])))
                self.io.append(
                    islamic_date(int(row[7]), int(row[8]), int(row[9])))
                self.bd.append(
                    bahai_date(int(row[10]), int(row[11]), int(row[12]),
                               int(row[13]), int(row[14])))
                self.bf.append(
                    bahai_date(int(row[15]), int(row[16]), int(row[17]),
                               int(row[18]), int(row[19])))
                self.mlc.append(
                    mayan_long_count_date(int(row[20]), int(row[21]),
                                           int(row[22]), int(row[23]),
                                           int(row[24])))
                self.mh.append(
                    mayan_haab_date(int(row[25]), int(row[26])))
                self.mt.append(
                    mayan_tzolkin_date(int(row[27]), int(row[28])))
                self.ax.append(
                    aztec_xihuitl_date(int(row[29]), int(row[30])))
                self.at.append(
                    aztec_tonalpohualli_date(int(row[31]), int(row[32])))

    class AppendixCTable2TestCase(AppendixCTable2TestCaseBase,
                                   unittest.TestCase):
        <test ethiopic appendix c 157>

        <test islamic appendix c 159>

        <test bahai appendix c 161>

        <test mayan appendix c 163>

```

This code is used in chunk 138.
Defines:

AppendixCTable2TestCase, never used.
AppendixCTable2TestCaseBase, used in chunks 4, 73, 79, 85, 117, 156, 158, 160, and 162.
Uses aztec_tonalpohualli_date 83, aztec_xihuitl_date 83, bahai_date 115, ethiopic_date 71,
islamic_date 77, mayan_haab_date 83, mayan_long_count_date 83, mayan_tzolkin_date 83, mt 92,
and rd 37.

```
156 <coptic and ethiopic calendars unit test 72>+≡
    class EthiopicAppendixCTestCase(AppendixCTable2TestCaseBase,
                                     unittest.TestCase):
        <test ethiopic appendix c 157>
```

This code is used in chunks 4 and 73.
Defines:
EthiopicAppendixCTestCase, never used.
Uses AppendixCTable2TestCaseBase 155.

```
157 <test ethiopic appendix c 157>≡
    def testEthiopic(self):
        for i in range(len(self.rd)):
            # ethiopic day
            self.assertEqual(ethiopic_from_fixed(self.rd[i]), self.ed[i])
            self.assertEqual(fixed_from_ethiopic(self.ed[i]), self.rd[i])
```

This code is used in chunks 155 and 156.
Defines:
testEthiopic, never used.
Uses ethiopic_from_fixed 71, fixed_from_ethiopic 71, and rd 37.

```
158 <islamic calendar unit test 78>+≡
    class IslamicAppendixCTestCase(AppendixCTable2TestCaseBase,
                                     unittest.TestCase):
        <test islamic appendix c 159>
```

This code is used in chunks 4 and 79.
Defines:
IslamicAppendixCTestCase, never used.
Uses AppendixCTable2TestCaseBase 155.

```
159 <test islamic appendix c 159>≡
    def testIslamic(self):
        for i in range(len(self.rd)):
            # islamic
            self.assertEqual(islamic_from_fixed(self.rd[i]), self.id[i])
            self.assertEqual(fixed_from_islamic(self.id[i]), self.rd[i])
            # islamic (observational)
            self.assertEqual(
                fixed_from_observational_islamic(self.io[i]), self.rd[i])
            self.assertEqual(
                observational_islamic_from_fixed(self.rd[i]), self.io[i])
```

This code is used in chunks 155 and 158.
Defines:
testIslamic, never used.
Uses fixed_from_islamic 77, fixed_from_observational_islamic 132, islamic_from_fixed 77,
observational_islamic_from_fixed 132, and rd 37.

```
160 <bahai calendar unit test 116>+≡
    class BahaiAppendixCTestCase(AppendixCTable2TestCaseBase,
                                     unittest.TestCase):
        <test bahai appendix c 161>
```

This code is used in chunks 4 and 117.
Defines:
BahaiAppendixCTestCase, never used.
Uses AppendixCTable2TestCaseBase 155.

```

161  <test bahai appendix c 161>≡
      def testBahai(self):
          for i in range(len(self.rd)):
              # bahai
              self.assertEqual(bahai_from_fixed(self.rd[i]), self.bd[i])
              self.assertEqual(fixed_from_bahai(self.bd[i]), self.rd[i])
              # bahai future
              self.assertEqual(future_bahai_from_fixed(self.rd[i]), self.bf[i])
              self.assertEqual(fixed_from_future_bahai(self.bf[i]), self.rd[i])

```

This code is used in chunks 155 and 160.

Defines:

testBahai, never used.

Uses bahai_from_fixed 115, fixed_from_bahai 115, fixed_from_future_bahai 115, future_bahai_from_fixed 115, and rd 37.

```

162  <mayan calendars unit test 84>+≡
      class MayanAppendixCTestCase(AppendixCTable2TestCaseBase,
                                   unittest.TestCase):
          <test mayan appendix c 163>

```

This code is used in chunks 4 and 85.

Defines:

MayanAppendixCTestCase, never used.

Uses AppendixCTable2TestCaseBase 155.

```

163  <test mayan appendix c 163>≡
      def testMayan(self):
          for i in range(len(self.rd)):
              # mayan (long count)
              self.assertEqual(
                  mayan_long_count_from_fixed(self.rd[i]), self.mlc[i])
              self.assertEqual(
                  fixed_from_mayan_long_count(self.mlc[i]), self.rd[i])
              # mayan (haab)
              self.assertEqual(mayan_haab_from_fixed(self.rd[i]), self.mh[i])
              # mayan (tzolkin)
              self.assertEqual(mayan_tzolkin_from_fixed(self.rd[i]), self.mt[i])

      def testAztec(self):
          for i in range(len(self.rd)):
              # aztec xihuitl
              self.assertEqual(aztec_xihuitl_from_fixed(self.rd[i]), self.ax[i])
              # aztec tonalpohualli
              self.assertEqual(
                  aztec_tonalpohualli_from_fixed(self.rd[i]), self.at[i])

```

This code is used in chunks 155 and 162.

Defines:

testAztec, never used.

testMayan, never used.

Uses aztec_tonalpohualli_from_fixed 83, aztec_xihuitl_from_fixed 83, fixed_from_mayan_long_count 83, mayan_haab_from_fixed 83, mayan_long_count_from_fixed 83, mayan_tzolkin_from_fixed 83, mt 92, and rd 37.

The third is for Hebrew (Standard and observational), Easter (Julian, Gregorian, Astronomical), Balinese Pawukon, Persian (Astronomical and arithmetic) and French Revolutionary (original and modified).

```

164 <appendixCTable3Tests 164>≡
    class AppendixCTable3TestCaseBase():

        def _toBoolean(self, c):
            return False if c == 'f' else True

        def setUp(self):
            reader = csv.reader(open("dates3.csv", "rU"),
                                delimiter=',',
                                quoting=csv.QUOTE_NONE)

            self.rd      = []
            self.hd      = [] # hebrew standard
            self.ho      = [] # hebrew observational
            self.je      = [] # julian easter
            self.ge      = [] # gregorian easter
            self.ae      = [] # astronomical easter
            self.bd      = [] # balinese pawukon
            self.pas     = [] # persian astronomical
            self.par     = [] # persian arithmetic
            self.fr      = [] # french original
            self.frm     = [] # french modified

            for row in reader:
                self.rd.append(int(row[0]))
                self.hd.append(hebrew_date(int(row[1]), int(row[2]), int(row[3])))
                self.ho.append(hebrew_date(int(row[4]), int(row[5]), int(row[6])))
                self.je.append(julian_date(int(row[7]), int(row[8]), int(row[9])))
                self.ge.append(gregorian_date(int(row[10]),
                                                int(row[11]),
                                                int(row[12])))
                self.ae.append(gregorian_date(int(row[13]),
                                                int(row[14]),
                                                int(row[15])))
                self.bd.append(balinese_date(self._toBoolean(row[16]),
                                                int(row[17]),
                                                int(row[18]),
                                                int(row[19]),
                                                int(row[20]),
                                                int(row[21]),
                                                int(row[22]),
                                                int(row[23]),
                                                int(row[24]),
                                                int(row[25])))
                self.pas.append(gregorian_date(int(row[26]),
                                                int(row[27]),
                                                int(row[28])))
                self.par.append(gregorian_date(int(row[29]),
                                                int(row[30]),
                                                int(row[31])))
                self.fr.append(french_date(int(row[32]),
                                                int(row[33]),
                                                int(row[34])))
                self.frm.append(french_date(int(row[35]),
                                                int(row[36]),
                                                int(row[37])))

    class AppendixCTable3TestCase(AppendixCTable3TestCaseBase,

```

```

                                unittest.TestCase):
    <test hebrew appendix c 166>

    <test easter appendix c 168>

    <test balinese appendix c 170>

    <test persian appendix c 172>

    <test french appendix c 174>

```

This code is used in chunk 138.

Defines:

AppendixCTable3TestCase, never used.

AppendixCTable3TestCaseBase, used in chunks 4, 76, 82, 91, 114, 122, 165, 167, 169, 171, and 173.

Uses balinese_date 89, easter 74, french_date 118, gregorian_date 52, hebrew_date 80, julian_date 59, and rd 37.

```

165 <hebrew calendar unit test 81>+≡
    class HebrewAppendixCTestCase(AppendixCTable3TestCaseBase,
                                    unittest.TestCase):
        <test hebrew appendix c 166>

```

This code is used in chunks 4 and 82.

Defines:

HebrewAppendixCTestCase, never used.

Uses AppendixCTable3TestCaseBase 164.

```

166 <test hebrew appendix c 166>≡
    def testHebrew(self):
        for i in range(len(self.rd)):
            self.assertEqual(fixed_from_hebrew(self.hd[i]), self.rd[i])
            self.assertEqual(hebrew_from_fixed(self.rd[i]), self.hd[i])
            # observational
            self.assertEqual(
                observational_hebrew_from_fixed(self.rd[i]), self.ho[i])
            self.assertEqual(
                fixed_from_observational_hebrew(self.ho[i]), self.rd[i])

```

This code is used in chunks 164 and 165.

Defines:

testHebrew, never used.

Uses fixed_from_hebrew 80, fixed_from_observational_hebrew 132, hebrew_from_fixed 80, observational_hebrew_from_fixed 132, and rd 37.

```

167 <ecclesiastical calendars unit test 75>+≡
    class EasterAppendixCTestCase(AppendixCTable3TestCaseBase,
                                    unittest.TestCase):
        <test easter appendix c 168>

```

This code is used in chunks 4 and 76.

Defines:

EasterAppendixCTestCase, never used.

Uses AppendixCTable3TestCaseBase 164.

```

168 <test easter appendix c 168>≡
    def testEaster(self):
        for i in range(len(self.rd)):
            self.assertEqual(
                gregorian_from_fixed(orthodox_easter(
                    gregorian_year_from_fixed(self.rd[i])),
                    self.je[i])
                self.assertEqual(
                    gregorian_from_fixed(alt_orthodox_easter(
                        gregorian_year_from_fixed(self.rd[i])),
                        self.je[i])
                    self.assertEqual(
                        gregorian_from_fixed(easter(
                            gregorian_year_from_fixed(self.rd[i])),
                            self.ge[i])
                        self.assertEqual(
                            gregorian_from_fixed(astronomical_easter(
                                gregorian_year_from_fixed(self.rd[i])),
                                self.ae[i])

```

This code is used in chunks 164 and 167.

Defines:

testEaster, never used.

Uses alt_orthodox_easter 74, astronomical_easter 132, easter 74, gregorian_from_fixed 56, gregorian_year_from_fixed 55, orthodox_easter 74, and rd 37.

```

169 <balinese calendar unit test 90>+≡
    class BalineseAppendixCTestCase(AppendixCTable3TestCaseBase):
        <test balinese appendix c 170>

```

This code is used in chunks 4 and 91.

Defines:

BalineseAppendixCTestCase, never used.

Uses AppendixCTable3TestCaseBase 164.

```

170 <test balinese appendix c 170>≡
    def testBalinese(self):
        for i in range(len(self.rd)):
            self.assertEqual(bali_pawukon_from_fixed(self.rd[i]), self.bd[i])

```

This code is used in chunks 164 and 169.

Defines:

testBalinese, never used.

Uses bali_pawukon_from_fixed 89 and rd 37.

```

171 <persian calendar unit test 113>+≡
    class PersianAppendixCTestCase(AppendixCTable3TestCaseBase,
                                    unittest.TestCase):
        <test persian appendix c 172>

```

This code is used in chunks 4 and 114.

Defines:

PersianAppendixCTestCase, never used.

Uses AppendixCTable3TestCaseBase 164.

```

172 <test persian appendix c 172>≡
    def testPersian(self):
        for i in range(len(self.rd)):
            # persian arithmetic
            self.assertEqual(
                fixed_from_arithmetic_persian(self.par[i]), self.rd[i])
            self.assertEqual(
                arithmetic_persian_from_fixed(self.rd[i]), self.par[i])
            # persian astronomical
            self.assertEqual(persian_from_fixed(self.rd[i]), self.pas[i])
            self.assertEqual(fixed_from_persian(self.pas[i]), self.rd[i])

```

This code is used in chunks 164 and 171.

Defines:

testPersian, never used.

Uses arithmetic_persian_from_fixed 112, fixed_from_arithmetic_persian 112, fixed_from_persian 112, persian_from_fixed 112, and rd 37.

```

173 <french revolutionary calendar unit test 119>+≡
    class FrenchRevolutionaryAppendixCTestCase(AppendixCTestCaseBase,
                                                unittest.TestCase):

        <test french appendix c 174>

```

This code is used in chunks 4 and 122.

Defines:

FrenchRevolutionaryAppendixCTestCase, never used.

Uses AppendixCTestCaseBase 164.

```

174 <test french appendix c 174>≡
    #def assertEquals(one, two, msg=""):
    #    print one, two

    def testFrenchRevolutionary(self):
        for i in range(len(self.rd)):
            # french revolutionary original
            self.assertEqual(fixed_from_french(self.fr[i]), self.rd[i])
            self.assertEqual(french_from_fixed(self.rd[i]), self.fr[i])
            # french revolutionary modified
            self.assertEqual(fixed_from_arithmetic_french(self.frm[i]), self.rd[i])
            self.assertEqual(arithmetic_french_from_fixed(self.rd[i]), self.frm[i])

```

This code is used in chunks 164 and 173.

Defines:

testFrenchRevolutionary, never used.

Uses arithmetic_french_from_fixed 120, fixed_from_arithmetic_french 120, fixed_from_french 120, french_from_fixed 120, and rd 37.

The fourth is for Chinese ...

```
175 <appendixCTable4Tests 175>≡
class AppendixCTable4TestCaseBase():

    def _toBoolean(self, c):
        return False if c == 'f' else True

    def setUp(self):
        reader = csv.reader(open("dates4.csv", "rU"),
                              delimiter=',',
                              quoting=csv.QUOTE_NONE)

        self.rd = []
        self.cd = [] # chinese date
        self.cn = [] # chinese day name
        self.ms = [] # major solar term
        self.ohs = [] # old hindu solar date
        self.mhs = [] # modern hindu solar date
        self.ahs = [] # astronomical hindu solar date
        self.ohl = [] # old hindu lunar date
        self.mhl = [] # modern hindu lunar date
        self.ahl = [] # astronomical hindu lunar date
        self.td = [] # tibetan date

        for row in reader:
            self.rd.append(int(row[0]))
            self.cd.append(chinese_date(int(row[1]), int(row[2]), int(row[3]),
                                         self._toBoolean(row[4]), int(row[5])))
            self.cn.append([int(row[6]), int(row[7])])
            self.ms.append(float(row[8]))
            self.ohs.append(hindu_solar_date(int(row[9]),
                                              int(row[10]),
                                              int(row[11])))
            self.mhs.append(hindu_solar_date(int(row[12]),
                                              int(row[13]),
                                              int(row[14])))
            self.ahs.append(hindu_solar_date(int(row[15]),
                                              int(row[16]),
                                              int(row[17])))
            self.ohl.append(old_hindu_lunar_date(int(row[18]),
                                                  int(row[19]),
                                                  self._toBoolean(row[20]),
                                                  int(row[21])))
            self.mhl.append(hindu_lunar_date(int(row[22]),
                                              int(row[23]),
                                              self._toBoolean(row[24]),
                                              int(row[25]),
                                              self._toBoolean(row[26])))
            self.ahl.append(hindu_lunar_date(int(row[27]),
                                              int(row[28]),
                                              self._toBoolean(row[29]),
                                              int(row[30]),
                                              self._toBoolean(row[31])))

            self.td.append(tibetan_date(int(row[32]),
                                         int(row[33]),
                                         self._toBoolean(row[34]),
                                         int(row[35]),
                                         self._toBoolean(row[36])))

class AppendixCTable4TestCase(AppendixCTable4TestCaseBase,
```

```

                                unittest.TestCase):
    <test chinese appendix c 177>
    <test old hindu appendix c 179>
    <test modern hindu appendix c 181>
    <test tibetan appendix c 183>

```

This code is used in chunk 138.

Defines:

AppendixCTable4TestCase, never used.

AppendixCTable4TestCaseBase, used in chunks 4, 88, 125, 128, 131, 176, 178, 180, and 182.

Uses chinese_date 123, hindu_lunar_date 126, hindu_solar_date 86, old_hindu_lunar_date 86, rd 37, and tibetan_date 129.

```

176 <chinese calendar unit test 124>+≡
    class ChineseAppendixCTestCase(AppendixCTable4TestCaseBase,
                                    unittest.TestCase):
        <test chinese appendix c 177>

```

This code is used in chunks 4 and 125.

Defines:

ChineseAppendixCTestCase, never used.

Uses AppendixCTable4TestCaseBase 175.

```

177 <test chinese appendix c 177>≡
    def testChinese(self):
        for i in range(len(self.rd)):
            self.assertEqual(fixed_from_chinese(self.cd[i]), self.rd[i])
            self.assertEqual(chinese_from_fixed(self.rd[i]), self.cd[i])
            self.assertEqual(chinese_day_name(self.rd[i]), self.cn[i])
            self.assertAlmostEqual(
                major_solar_term_on_or_after(self.rd[i]), self.ms[i], 6)

```

This code is used in chunks 175 and 176.

Defines:

testChinese, never used.

Uses chinese_day_name 123, chinese_from_fixed 123, fixed_from_chinese 123, major_solar_term_on_or_after 123, and rd 37.

```

178 <old hindu calendars unit test 87>+≡
    class OldHinduAppendixCTestCase(AppendixCTable4TestCaseBase,
                                    unittest.TestCase):
        <test old hindu appendix c 179>

```

This code is used in chunks 4 and 88.

Defines:

OldHinduAppendixCTestCase, never used.

Uses AppendixCTable4TestCaseBase 175.

```

179 <test old hindu appendix c 179>≡
    def testOldHindu(self):
        for i in range(len(self.rd)):
            # solar
            self.assertEqual(fixed_from_old_hindu_solar(self.ohs[i]), self.rd[i])
            self.assertEqual(old_hindu_solar_from_fixed(self.rd[i]), self.ohs[i])
            # lunisolar
            self.assertEqual(fixed_from_old_hindu_lunar(self.ohl[i]), self.rd[i])
            self.assertEqual(old_hindu_lunar_from_fixed(self.rd[i]), self.ohl[i])

```

This code is used in chunks 175 and 178.

Defines:

testOldHindu, never used.

Uses fixed_from_old_hindu_lunar 86, fixed_from_old_hindu_solar 86, old_hindu_lunar_from_fixed 86, old_hindu_solar_from_fixed 86, and rd 37.

```
180  <modern hindu calendars unit test 127>+≡  
      class ModernHinduAppendixCTestCase(AppendixCTable4TestCaseBase,  
                                          unittest.TestCase):  
          <test modern hindu appendix c 181>
```

This code is used in chunks 4 and 128.

Defines:

ModernHinduAppendixCTestCase, never used.

Uses AppendixCTable4TestCaseBase 175.

```

181 <test modern hindu appendix c 181>≡
def testHinduSolarModernToFixed(self):
    for i in range(len(self.rd)):
        # hindu solar
        # modern
        self.assertEqual(fixed_from_hindu_solar(self.mhs[i]), self.rd[i])

def testHinduSolarModernFromFixed(self):
    for i in range(len(self.rd)):
        # hindu solar
        # modern
        self.assertEqual(hindu_solar_from_fixed(self.rd[i]), self.mhs[i])

def testHinduSolarAstronomicalToFixed(self):
    for i in range(len(self.rd)):
        # astronomical
        self.assertEqual(fixed_from_astro_hindu_solar(self.ahs[i]), self.rd[i])

def testHinduSolarAstronomicalFromFixed(self):
    for i in range(len(self.rd)):
        # astronomical
        self.assertEqual(astro_hindu_solar_from_fixed(self.rd[i]), self.ahs[i])

def testHinduLunisolarModernToFixed(self):
    for i in range(len(self.rd)):
        # hindu lunisolar
        # modern
        self.assertEqual(fixed_from_hindu_lunar(self.mhl[i]), self.rd[i])

def testHinduLunisolarModernFromFixed(self):
    for i in range(len(self.rd)):
        # hindu lunisolar
        # modern
        self.assertEqual(hindu_lunar_from_fixed(self.rd[i]), self.mhl[i])

def testHinduLunisolarAstronomicalToFixed(self):
    for i in range(len(self.rd)):
        # hindu lunisolar
        # astronomical
        self.assertEqual(fixed_from_astro_hindu_lunar(self.ahl[i]), self.rd[i])

def testHinduLunisolarAstronomicalFromFixed(self):
    for i in range(len(self.rd)):
        # hindu lunisolar
        # astronomical
        self.assertEqual(astro_hindu_lunar_from_fixed(self.rd[i]), self.ahl[i])

```

This code is used in chunks 175 and 180.

Defines:

```

testHinduLunisolarAstronomicalFromFixed, never used.
testHinduLunisolarAstronomicalToFixed, never used.
testHinduLunisolarModernFromFixed, never used.
testHinduLunisolarModernToFixed, never used.
testHinduSolarAstronomicalFromFixed, never used.
testHinduSolarAstronomicalToFixed, never used.
testHinduSolarModernFromFixed, never used.
testHinduSolarModernToFixed, never used.

```

Uses astro_hindu_lunar_from_fixed 126, astro_hindu_solar_from_fixed 126,
fixed_from_astro_hindu_lunar 126, fixed_from_astro_hindu_solar 126, fixed_from_hindu_lunar
126, fixed_from_hindu_solar 126, hindu_lunar_from_fixed 126, hindu_solar_from_fixed 126,
and rd 37.


```

182  <tibetan calendar unit test 130>+≡
      class TibetanAppendixCTestCase(AppendixCTable4TestCaseBase, unittest.TestCase):
          <test tibetan appendix c 183>

```

This code is used in chunks 4 and 131.

Defines:

TibetanAppendixCTestCase, never used.

Uses AppendixCTable4TestCaseBase 175.

```

183  <test tibetan appendix c 183>≡
      def testTibetan(self):
          for i in range(len(self.rd)):
              self.assertEqual(fixed_from_tibetan(self.td[i]), self.rd[i])
              self.assertEqual(tibetan_from_fixed(self.rd[i]), self.td[i])

```

This code is used in chunks 175 and 182.

Defines:

testTibetan, never used.

Uses fixed_from_tibetan 129, rd 37, and tibetan_from_fixed 129.

The fifth is for Astronomical calculations

```

184 <appendixCTable5Tests 184>≡
    class AppendixCTable5TestCaseBase():

        def _toBoolean(self, c):
            return False if c == 'f' else True

        def setUp(self):
            reader = csv.reader(open("dates5.csv", "rU"),
                                delimiter=',',
                                quoting=csv.QUOTE_NONE)

            self.rd = []
            self.sl = [] # solar longitude
            self.nse = [] # next solstice/equinox
            self.ll = [] # lunar longitude
            self.nnm = [] # next new moon
            self.dip = [] # dawn in Paris
            self.sij = [] # sunset in Jerusalem

            for row in reader:
                self.rd.append(int(row[0]))
                self.sl.append(float(row[1]))
                self.nse.append(float(row[2]))
                self.ll.append(float(row[3]))
                #self.nnm.append(float(row[4])) # read from errata file
                self.dip.append(row[5])
                self.sij.append(row[9])

            reader1 = csv.reader(open("dates5.errata.csv", "rU"),
                                delimiter=',',
                                quoting=csv.QUOTE_NONE)

            for row in reader1:
                self.nnm.append(float(row[1]))

    class AppendixCTable5TestCase(AppendixCTable5TestCaseBase,
                                   unittest.TestCase):
        <test astronomy appendix c 186>

```

This code is used in chunk 138.

Defines:

AppendixCTable5TestCase, never used.

AppendixCTable5TestCaseBase, used in chunks 4, 111, 134, and 185.

Uses dawn 92, longitude 92, next 16, rd 37, and sunset 92.

```

185 <time and astronomy unit test 94>+≡
    class AstronomyAppendixCTestCase(AppendixCTable5TestCaseBase,
                                       unittest.TestCase):
        <test astronomy appendix c 186>

```

This code is used in chunks 4 and 111.

Defines:

AstronomyAppendixCTestCase, never used.

Uses AppendixCTable5TestCaseBase 184.

```

186 <test astronomy appendix c 186>≡
def testSolarLongitude(self):
    for i in range(len(self.rd)):
        # +0.5 takes into account that the value has to be
        # calculated at 12:00 UTC
        self.assertAlmostEqual(solar_longitude(self.rd[i] + 0.5),
                                self.sl[i],
                                6)

def testNextSolsticeEquinox(self):
    # I run some tests for Gregorian year 1995 about new Moon and
    # start of season against data from HM Observatory...and they
    # are ok
    for i in range(len(self.rd)):
        t = [solar_longitude_after(SPRING, self.rd[i]),
              solar_longitude_after(SUMMER, self.rd[i]),
              solar_longitude_after(AUTUMN, self.rd[i]),
              solar_longitude_after(WINTER, self.rd[i])]
        self.assertAlmostEqual(min(t), self.nse[i], 6)

def testLunarLongitude(self):
    for i in range(len(self.rd)):
        self.assertAlmostEqual(lunar_longitude(self.rd[i]),
                                self.ll[i],
                                6)

def testNextNewMoon(self):
    for i in range(len(self.rd)):
        self.assertAlmostEqual(new_moon_at_or_after(self.rd[i]),
                                self.nnm[i],
                                6)

def testDawnInParis(self):
    # as clarified by Prof. Reingold in CL it is:
    # (dawn day paris 18d0)
    # note that d0 stands for double float precision and in
    # the Python routines we use mpf with 52 digits for dawn()
    alpha = angle(18, 0, 0)
    for i in range(len(self.rd)):
        if (self.dip[i] == BOGUS):
            self.assertEqual(dawn(self.rd[i], PARIS, alpha), self.dip[i])
        else:
            self.assertAlmostEqual(
                mod(dawn(self.rd[i], PARIS, alpha), 1),
                mpf(self.dip[i]),
                6)

def testSunsetInJerusalem(self):
    for i in range(len(self.rd)):
        self.assertAlmostEqual(
            mod(sunset(self.rd[i], JERUSALEM), 1),
            float(self.sij[i]),
            6)

```

This code is used in chunks 184 and 185.

Defines:

```

testDawnInParis, never used.
testLunarLongitude, never used.
testNextNewMoon, never used.
testNextSolsticeEquinox, never used.
testSolarLongitude, never used.
testSunsetInJerusalem, never used.

```

Uses `angle 92`, `AUTUMN 99`, `BOGUS 13`, `dawn 92`, `JERUSALEM 92 132`, `lunar_longitude 109`, `mod 15`, `new_moon_at_or_after 109`, `PARIS 118`, `rd 37`, `solar_longitude 97`, `solar_longitude_after 99`, `SPRING 99`, `start 47`, `SUMMER 99`, `sunset 92`, and `WINTER 99`.

2.23 Test Coverage

I want to understand what I am covering with my tests. This is have a quantitative evaluation of the amount of lines of code of my software I am traversing with my tests. This does not give me any confidence about the correctness of my code, but at least it says what is lacking a test.

I found a nice, simple tool written in PYTHON, coverage [9]. I defined a new target in Makefile to produce an HTML report

```
187 <Makefile: coverage target 187>≡
    .PHONY : coverage
    coverage: pycalcaltests.py appendixCUnitTest.py testdata
        coverage -e -x pycalcaltests.py      # -e drop previous
        coverage -b -i -d html pycalcal.py # HTML report for pycalcal.py
```

This code is used in chunk 210.

2.24 Cross checking

The following scripts have been used to generate and check the initial set of function signatures for the project.

2.24.1 Generating all function signatures

The follwing command will generate all PYTHON function signatures from COMMON LISP counterparts:

```
188 <extractcc3signatures 188>≡
    # <generated code warning 1>
    cat calendrica-3.0.cl | grep -e '^ (defun ' | \
    sed \
        -e 's/ (/(/g' \
        -e 's/-/_/g' \
        -e 's/).*$/):/g' \
        -e 's/ /, /g' \
        -e 's/(defun,/def/' \
        -e 's/lambda/lam/g' \
        -e 's/\([a-zA-z_][a-zA-Z_]*\)?(/is_\1(/g' \
        -e 's/?//g' | \
    tr -d '\15\32'
```

Root chunk (not used in this document).

and here is the one to get the ones from pycalcal.py:

```
189 <extractcalcalsignatures 189>≡
    cat pycalcal.py | grep '^def '
```

Root chunk (not used in this document).

The idea is to extract all functions defined in calendrica-3.0.cl using extractcc3signatures and compare them with the ones in pycalcal.py like:

```
prompt> cat calendrica-3.0.cl | extractcc3signatures | sort > allcc3
prompt> cat pycalcal.py | extractcalcalsignatures | sort > allcalcal
prompt> diff allcc3 allcalcal
```

There is also the need to check all constants, i.e. defconstant. TODO.

2.24.2 Checking the function signatures

```
190 <Makefile: cross checks 190>≡
    .PHONY : missing
    missing: extractcc3signatures extractcalcalsignatures
        - cat calendrica-3.0.cl | extractcc3signatures - | sort > /tmp/allcc3
        - cat $(NW_MAIN:.nw=.py) | extractcalcalsignatures - | sort > /tmp/allcalcal
        - diff /tmp/allcc3 /tmp/allcalcal

    extractcc3signatures: $(NW_MAIN:.nw=.py)
        notangle -Rextractcc3signatures $(NW_MAIN) $(CPIF) extractcc3signatures
        -chmod ug+x extractcc3signatures

    extractcalcalsignatures: $(NW_MAIN:.nw=.py)
        notangle -Rextractcalcalsignatures $(NW_MAIN) $(CPIF) extractcalcalsignatures
        -chmod ug+x extractcalcalsignatures
```

This code is used in chunk 208.

Chapter 3

Future evolutions

I would like to define an OO version of these calendrical algorithms. This can be implemented on top of existing functional implementation, i.e. `GregorianCalendar` will have relevant fields like, day, month and year and methods like `isEaster()` or so.

Validation of astronomical algorithm could be performed using data service for astronomical applications as made available by the US Naval Observatory web services [12] or the IMCCE [5].

I created a google app for `pymathcal`. In a directory named 'calendrica' I did put `calendrica.py` and `app.yaml`. I also create a 'lib' directory where I copied the whole `mpmath` distribution. You can run in web app in your development environment using

```
dev_appserver.py calendrica/
```

and browsing it at

```
http://localhost:8080/
```

Once ready you can deploy it using

```
appcfg.py update calendrica/
```

```
191 <calendrica.py 191>≡
    # <generated code warning 1>
    #####
    # This is a Google App that converts a date from/to a limited #
    # set (gregorian, hebrew, islamic) calendars using pymathcal #
    #                                                                 #
    # It assumes the app has a lib dir with the following pkgs:    #
    #   * pymathcal.py                                             #
    #   * mpmath package <http://code.google.com/p/mpmath/>        #
    #     (the full 'mpmath' dir of the installation under         #
    #     .../Python25/Lib/site-packages/mpmath)                  #
    # This file, calendrica.py, and its companion configuration   #
    # app.yaml are contained in a directory named calendrica.     #
    #####
    from google.appengine.ext import webapp
    from google.appengine.ext.webapp.util import run_wsgi_app
    import cgi

    # PyCalCal imports
    import os
    import sys
    sys.path.reverse()
    sys.path.append(os.path.abspath('./lib'))
    sys.path.reverse()
    import pymathcal as cal
```

```

class MainPage(webapp.RequestHandler):
    def get(self):
        self.response.out.write("""
        <html>
        <body>
        <div>This is a simple web app which allows to
        convert dates between calendars.<br />
        This is just the backbone using
        <a href="http://code.google.com/p/pycalcal/">pycalcal</a>.<br />
        Try it out with input like <code>[2009, 10, 25]</code>
        for a Gregorian date.
        </div>
        <form action="/convert" method="get">
        <div><input type="text" name="date" /></div>
        <div>
        <select name="fromCalendar">
        <option value="gregorian" selected="selected">
        Gregorian date (year, month, day) [int, int, int]</option>
        <option value="islamic">
        Islamic date (year, month, day) [int, int, int]</option>
        <option value="observational_islamic">
        Observational Islamic date (year, month, day) [int, int, int]
        </option>
        <option value="hebrew">
        Hebrew date (year, month, day) [int, int, int]</option>
        <option value="observational_hebrew">
        Observational Hebrew date (year, month, day) [int, int, int]
        </option>
        <option value="egyptian">
        Egyptian date (year, month, day) [int, int, int]</option>
        <option value="armenian">
        Armenian date (year, month, day) [int, int, int]</option>
        <option value="iso">
        ISO date (year, week, day) [int, int, int]</option>
        <option value="coptic">
        Coptic date (year, month, day) [int, int, int]</option>
        <option value="ethiopic">
        Ethiopic date (year, month, day) [int, int, int]</option>
        <option value="mayan_long_count">
        Mayan Long Count date (baktun, katun, tun, uinal, kin)
        [int, int, int, int, int]</option>
        <option value="old_hindu_solar">
        Old Hindu solar date (year, month, day) [int, int, int]</option>
        <option value="hindu_solar">
        Hindu solar date (year, month, day) [int, int, int]</option>
        <option value="astro_hindu_solar">
        Astro Hindu solar date (year, month, day) [int, int, int]
        </option>
        <option value="old_hindu_lunar">
        Old Hindu lunar date (year, month, leap, day)
        [int, int, bool, int]</option>
        <option value="hindu_lunar">
        Hindu lunar date (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        <option value="astro_hindu_lunar">
        Astro Hindu lunar date (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        <option value="tibetan">Tibetan date
        (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        """)

```

```

<option value="persian">
Persian date (year, month, day) [int, int, int]</option>
<option value="arithmetic_persian">
Arithmetic Persian date (year, month, day)
[int, int, int]</option>
<option value="bahai">
Bahai date (major, cycle, year, month, day)
[int, int, int, int, int]</option>
<option value="future_bahai">
Future Bahai date (major, cycle, year, month, day)
[int, int, int, int, int]</option>
<option value="french">
French date (year, month, day) [int, int, int]</option>
<option value="arithmetic_french">
Arithmetic French date (year, month, day) [int, int, int]
</option>
<option value="chinese">
Chinese date (cycle, year, month, leap_month, day)
[int, int, int, bool, int]</option>
<option value="julian">
Julian date (year, month, day) [int, int, int]</option>
<option value="roman">
Roman date (year, month, event, count, leap)
[int, int, int, int, bool]</option>
<option value="jd">Julian Day int</option>
<option value="mjd">Modified Julian Day int</option>
<option value="moment">Moment float</option>
</select>
</div>
<div>
<select name="toCalendar">
<option value="gregorian">
Gregorian date (year, month, day) [int, int, int]</option>
<option value="islamic">
Islamic date (year, month, day) [int, int, int]</option>
<option value="observational_islamic">
Observational Islamic date (year, month, day)
[int, int, int]</option>
<option value="hebrew">
Hebrew date (year, month, day) [int, int, int]</option>
<option value="observational_hebrew">
Observational Hebrew date (year, month, day)
[int, int, int]</option>
<option value="egyptian">
Egyptian date (year, month, day) [int, int, int]</option>
<option value="armenian">
Armenian date (year, month, day) [int, int, int]</option>
<option value="iso" selected="selected">
ISO date (year, week, day) [int, int, int]</option>
<option value="coptic">
Coptic date (year, month, day) [int, int, int]</option>
<option value="ethiopic">
Ethiopic date (year, month, day) [int, int, int]</option>
<option value="mayan_long_count">
Mayan Long Count date (baktun, katun, tun, uinal, kin)
[int, int, int, int, int]</option>
<option value="old_hindu_solar">
Old Hindu solar date (year, month, day)
[int, int, int]</option>
<option value="hindu_solar">
Hindu solar date (year, month, day) [int, int, int]</option>

```



```

        <option value="astro_hindu_solar">
        Astro Hindu solar date (year, month, day)
        [int, int, int]</option>
        <option value="old_hindu_lunar">
        Old Hindu lunar date (year, month, leap, day)
        [int, int, bool, int]</option>
        <option value="hindu_lunar">
        Hindu lunar date (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        <option value="astro_hindu_lunar">
        Astro Hindu lunar date (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        <option value="tibetan">
        Tibetan date (year, month, leap_month, day, leap_day)
        [int, int, bool, int, bool]</option>
        <option value="persian">
        Persian date (year, month, day) [int, int, int]</option>
        <option value="arithmetic_persian">
        Arithmetic Persian date (year, month, day)
        [int, int, int]</option>
        <option value="bahai">
        Bahai date (major, cycle, year, month, day)
        [int, int, int, int, int]</option>
        <option value="future_bahai">
        Future Bahai date (major, cycle, year, month, day)
        [int, int, int, int, int]</option>
        <option value="french">
        French date (year, month, day) [int, int, int]</option>
        <option value="arithmetic_french">
        Arithmetic French date (year, month, day)
        [int, int, int]</option>
        <option value="chinese">
        Chinese date (cycle, year, month, leap_month, day)
        [int, int, int, bool, int]</option>
        <option value="julian">
        Julian date (year, month, day) [int, int, int]</option>
        <option value="roman">
        Roman date (year, month, event, count, leap)
        [int, int, int, int, bool]</option>
        <option value="jd">Julian Day int</option>
        <option value="mjd">Modified Julian Day int</option>
        <option value="moment">Moment float</option>
    </select>
</div>
<div><input type="submit" value="Convert"></div>
</form>
</body>
</html>""")

```

```

class Converter(webapp.RequestHandler):
    def get(self):
        values = dict((k, cgi.escape(self.request.get(k))) for k in (
            'fromCalendar',
            'toCalendar',
            'date'))

        fixed_from_FROM_name = 'fixed_from_' + values['fromCalendar']
        TO_from_fixed_name = values['toCalendar'] + '_from_fixed'
        from_method = cal.__dict__[fixed_from_FROM_name]
        to_method = cal.__dict__[TO_from_fixed_name]

```

```

date = eval(values['date'])
result = to_method(from_method(date))

self.response.out.write('<html><body>You converted ' +
    values['fromCalendar'] +
    ' date ' +
    values['date'] +
    ' to ' +
    values['toCalendar'] +
    ' date ' +
    repr(result)
    )
self.response.out.write('</pre><br />
<a href="/">Try again!</a></body></html>')

application = webapp.WSGIApplication(
    [('/', MainPage),
     ('/convert', Converter)],
    debug=True)

#
# the code to randomly generate from/to functions could come handy some day
#
# class OldMainPage(webapp.RequestHandler):
#     def get(self):
#         self.response.headers['Content-Type'] = 'text/plain'
#         self.response.out.write(self.doUsefulComputation())

#     def doUsefulComputation(self):
#         """Convert from a Gregorian date to a random calendar from the ones
#         available in pycalcal"""
#         b = cal.fixed_from_gregorian(cal.gregorian_date(1967, 1, 30))

#         #####
#         # pick a random calendar
#         #####
#         from random import choice
#         import re
#         # here is the 'xyz_from_fixed' string
#         f = choice(self.fromFixed())
#         # invoke the conversion to destination calendar
#         d = cal.__dict__[f](b)

#         # get the calendar name: 'xyz'
#         r = re.compile('(.+)(_from_fixed)')
#         n = r.match(f).group(1)
#         res = 'gergorian=1967, 1, 30; %s=%s' % (n, d)

#         return res

#     def fromFixed(self):
#         import re
#         m2 = re.compile('._from_fixed$')
#         m3 = re.compile('._(year|week|day|year_bearer)_from_fixed$')
#         l_fromFixed = []
#         for i in dir(cal):
#             if re.match(m2, i) and not re.match(m3, i):

```

```

#             l_fromFixed.append(i)
#         return l_fromFixed

#     def toFixed(self):
#         import re
#         m1 = re.compile('^fixed_from_')
#         l_toFixed = []
#         for i in dir(cal):
#             if re.match(m1, i):
#                 l_toFixed.append(i)
#         return l_toFixed

```

```

def main():
    run_wsgi_app(application)

if __name__ == "__main__":
    main()

```

Root chunk (not used in this document).
 Uses `fixed_from_gregorian 55` and `gregorian_date 52`.

This is the config file

```

192 <app.yaml 192>≡
    application: calendrica
    version: 1
    runtime: python
    api_version: 1

    handlers:
    - url: /. *
      script: calendrica.py

```

Root chunk (not used in this document).

Chapter 4

Technicalities

By definition of *Calendrical Calculations*'s authors, the only authoritative document is their COMMON LISP code. For this reason I want to keep track of the relevant COMMON LISP code when defining my PYTHON counterpart and the following sections explain how I make sure I implement all the algorithms and how I pair COMMON LISP code and PYTHON code..

4.1 Inserting snippets of Common LISP code

I would like to be able to prepend the relevant snippet of (PYTHON-commented) COMMON LISP code from `calendrica-3.0.cl` to the relevant PYTHON counterpart.

I created a sort of pre-processor for NOWEB: it allows to write a code chunk reference like `# see lines x-y in file` and to obtain the insertion of a portion of a file. Its semantic is: "Insert the content of file 'file' from line 'x' to line 'y'".

The preprocessor is called `premarkup` and is defined as follows:

```
193 <premarkup 193>≡
    #! /bin/awk -f
    # <generated code warning 1>
    #-*- mode: awk -*-
    # substitute chunks of the form 'see lines <x>-<y> in <file>'
    # with the contents of file 'file' from line 'x' to line 'y' inclusive
    # eventually prefixed

    /# see lines [0-9][0-9]*-[0-9][0-9]* in .*/ {
        # extract x and y
        # (file -> substr($1,3))
        split($4, a, "-")
        # the command 'excerpt 10 20 file'
        # prints the contents of file 'file' from line 10 till 20
        system("excerpt " a[1] " " a[2] " " $6 " | prefix")

        # move to next line
        next}

    # match and print all the rest
    1
```

Root chunk (not used in this document).
Uses `next` 16.

Note that it uses other two scripts, `prefix` and `excerpt`, to insert and comment out the relevant snippet of code.

```
194 <prefix 194>≡
    #!/bin/sh
    # <generated code warning 1>
    # -*- mode: shell -*-
    scriptname=$0

    function usage () {
        cat <EOF
Usage: ${scriptname} [-p prefix] [prepend]
    -p prefix    define new prefix string (default is '# ')
    -h           displays basic help
Prepend <prepend><prefix> to each line.
EOF
        exit 0
    }

    pref='# '
    while getopts hp: name
    do
        case $name in
            p)    pref="$OPTARG";;
            h)    usage;;
            esac
        done

        shift $((($OPTIND - 1))
        if (( $# > 1 ))
        then
            return 1
        fi

        blanks="$*"

        sed -e "s,[^.]},${blanks}${pref}&,"
```

Root chunk (not used in this document).

And here are the targets for Makefile:

```
195 <Makefile: premarkup 195>≡
    premarkup: prefix excerpt
                notangle -Rpremarkup $(NW_MAIN) $(CPIF) premarkup
                -chmod ug+x premarkup
```

This code is used in chunk 232.

```
196 <Makefile: prefix 196>≡
    prefix:
                notangle -Rprefix $(NW_MAIN) $(CPIF) prefix
                -chmod ug+x prefix
```

This code is used in chunk 232.

Excerpt, a simple invocation of sed:

```
197 <excerpt 197>≡
    #!/bin/sh
    # <generated code warning 1>

    # excerpt b e f
    # print contents of file 'f' from line 'b' till 'e'

    sed -n $1,$2p $3
    # $ this is to fake emacs in nw source
```

Root chunk (not used in this document).

An example of use of premarkup is as follows:

```
prompt> premarkup pycalcal.nw | noweave -n -delay - | cpif pycalcal.tex
```

The target for Makefile is

```
198 <Makefile: excerpt 198>≡
    excerpt:
        notangle -Rexcerpt $(NW_MAIN) $(CPIF) excerpt
        -chmod ug+x excerpt
```

This code is used in chunk 232.

4.2 How to avoid Noweb from indexing comments: HACK?

The problem with NOWEB indexing is that comments are parsed as well and I do not like to get index entries in COMMON LISP code. One possible hack is to create indexing info from a copy of the L^AT_EX doc where COMMON LISP code is blanked out. So blankexcerpt will try to accomplish this:

```
199 <blankexcerpt 199>≡
    #!/bin/sh

    # blankexcerpt b e f
    # print contents of file 'f' from line 'b' till 'e' and changing
    # each char into a blank
    #excerpt $1 $2 $3 | sed -e's/./ /g'

    # In fact we want e - b + 1 blank lines!!!!
    for i in $(seq $1..$2)
    do
        echo ""
    done

    # $ this is to fake emacs in nw source
```

Root chunk (not used in this document).

Still I am not sure how to use this...but we probably need something like:

```
200 <blankpremarkup 200>≡
    #! /bin/awk -f
    # substitute chunks of the form '# see lines <x>-<y> in <file>'
    # with the contents of file 'file' from line 'x' to line 'y' inclusive
    # eventually prefixed

    /* see lines [0-9][0-9]*-[0-9][0-9]* in .*/ {
        # extract x and y
        # (file -> substr($1,3))
        split($4, a, "-")
        # the command 'blankexcerpt 10 20 file'
        # prints the contents of file 'file' from line 10 till 20
        system("blankexcerpt " a[1] " " a[2] " " $6 " | prefix")

        # move to next line
        next}

    #$ this is to fake emacs in nw source
    # match and print all the rest
    1
```

Root chunk (not used in this document).

Uses [next 16](#).

The relevant Makefile targets are:

```
201 <Makefile: blankpremarkup 201>≡
    blankpremarkup: prefix blankexcerpt
        notangle -Rblankpremarkup $(NW_MAIN) $(CPIF) blankpremarkup
        -chmod ug+x blankpremarkup
```

This code is used in chunk [232](#).

```
202 <Makefile: blankexcerpt 202>≡
    blankexcerpt:
        notangle -Rblankexcerpt $(NW_MAIN) $(CPIF) blankexcerpt
        -chmod ug+x blankexcerpt
```

This code is used in chunk [232](#).

4.3 Make It Work

The Makefile I defined is minimalistically simple.

```
203 <Makefile 203>≡
    # <generated code warning 1>
    <Makefile: heading comment 204>
    VERSION = <project version 10>
    <Makefile: files declarations 205>
    <Makefile: commands 206>
    <Makefile: suffixes 207>
    <Makefile: targets 208>
```

Root chunk (not used in this document).

```

204  <Makefile: heading comment 204>≡
      #####
      # Author: Enrico Spinielli
      #
      # Makefile for my noweb project on calendrical calculations in Python.
      # Got hints from Makefile in Noweb distribution
      #
      #####

```

This code is used in chunk 203.

```

205  <Makefile: files declarations 205>≡
      NW_MAIN=pycalcal.nw
      NW_SRC=
      PYTHON_SITE_PACKAGES=/Library/Python/2.6/site-packages
      UNIT_TEST_FILES=basicCodeUnitTest.py \
                      egyptianAndArmenianCalendarsUnitTest.py \
                      gregorianCalendarUnitTest.py \
                      isoCalendarUnitTest.py \
                      julianCalendarUnitTest.py \
                      copticAndEthiopicCalendarsUnitTest.py \
                      ecclesiasticalCalendarsUnitTest.py \
                      islamicCalendarUnitTest.py \
                      hebrewCalendarUnitTest.py \
                      mayanCalendarsUnitTest.py \
                      oldHinduCalendarsUnitTest.py \
                      balineseCalendarUnitTest.py \
                      timeAndAstronomyUnitTest.py \
                      persianCalendarUnitTest.py \
                      bahaiCalendarUnitTest.py \
                      frenchRevolutionaryCalendarUnitTest.py \
                      chineseCalendarUnitTest.py \
                      modernHinduCalendarsUnitTest.py \
                      tibetanCalendarUnitTest.py \
                      astronomicalLunarCalendarsUnitTest.py

```

This code is used in chunk 203.

```

206  <Makefile: commands 206>≡
      # We can use:
      # PREMARKUP=./premarkup $(NW_MAIN)
      # PREMARKUP=./blankpremarkup $(NW_MAIN)
      PREMARKUP=cat $(NW_MAIN)

      NOTANGLE_PURE=cat $(NW_MAIN) | notangle
      NOTANGLE=$(PREMARKUP) | notangle
      NOWEAVE=$(PREMARKUP) | noweave -n -delay
      NODEFS=$(PREMARKUP) | nodefs
      #LATEX=pdflatex -include-directory=$(cygpath -w /usr/local/noweb/texmf)
      LATEX=latex
      PDFLATEX=pdflatex
      BIBTEX=bibtex

      # to be used only when there are multiple .nw files
      NOINDEX=noindex
      # change to ">" to ensure all sources are always made
      CPIF=| cpif

      NW_ALL=$(NW_MAIN) $(NW_SRC)

```

This code is used in chunk 203.


```

207  <Makefile: suffixes 207>≡
    .SUFFIXES:
    .SUFFIXES: .nw .tex .py .dvi .defs .html .pdf

    .nw.py:
        $(NOTANGLE) -filter btdefn -R$*.py - $(CPIF) $*.py

    .nw.html:
        $(NOWEAVE) -filter l2h -filter btdefn -index -html $*.nw $(CPIF) $*.html

    .nw.defs:
        $(NODEFS) - $(CPIF) $*.defs

    .nw.tex:
        $(NOWEAVE) -filter btdefn -index - $(CPIF) $*.tex

    .tex.dvi:
        $(LATEX) $*; \
        $(NOINDEX) $*; \
        if grep -s 'There were undefined references' $*.log; \
        then $(BIBTEX) $*; fi; \
        while grep -s 'Rerun to get cross-references right' $*.log; \
        do $(LATEX) $*; \
        done; \

    .tex.pdf:
        $(PDFLATEX) $*; \
        $(NOINDEX) $*; \
        if grep -s 'There were undefined references' $*.log; \
        then $(BIBTEX) $*; fi; \
        while grep -s 'Rerun to get cross-references right' $*.log; \
        do $(PDFLATEX) $*; \
        done; \

```

This code is used in chunk 203.

```

208 <Makefile: targets 208>≡
# %.chk (this is a check for latex)
# Do not delete the following targets:
.PRECIOUS: %.aux %.bbl

.PHONY : all
all: figure pycalcal.pdf $(NW_MAIN:.nw=.py) pycalcaltests.py

.PHONY : index
pycalcal.defs: $(NW_MAIN) premarkup
    $(NODEFS) - $(CPIF) $*.defs

#all.defs: pycalcal.defs
#    cat pycalcal.defs ${CPIF} all.defs
#    sort -u $(NW_ALL:.nw=.defs) $(CPIF) all.defs

# TeX (use predefined rule)
pycalcal.tex: premarkup

pycalcal.pdf: $(NW_MAIN:.nw=.tex) figure

# Python files (for pycalcal.py use predefined rule +
#    extra dependency on premarkup)
pycalcal.py: premarkup

.PHONY : figure

figure:
    mptopdf figure.mp

<Makefile: distro 209>
<Makefile: unit tests and targets 210>
<Makefile: cross checks 190>
<Makefile: tools 232>
<Makefile: clean 233>
.PHONY : webapp
webapp: calendrica.py
    -mkdir -p calendrica/lib
    -cp pycalcal.py calendrica/lib
    -cp -fR $(PYTHON_SITE_PACKAGES)/mpmath calendrica/lib

calendrica.py: $(NW_MAIN:.nw=.py)
    -mkdir calendrica
    $(NOTANGLE) -filter btdefn -R$.py - $(CPIF) calendrica/calendrica.py
    $(NOTANGLE) -filter btdefn -Rapp.yaml - $(CPIF) calendrica/app.yaml

```

This code is used in chunk 203.

```

209  <Makefile: distro 209>≡
      DISTRO_FILES=$(NW_MAIN) $(NW_MAIN:.nw=.py) $(NW_MAIN:.nw=.pdf) \
          README \
          INSTALL \
          STATUS \
          COPYRIGHT_DERSHOWITZ_REINGOLD \
          makemake.sh \
          figure.mp \
          astro.mp

      .PHONY : distro
      distro: all
          tar -czf pycalcal_$$$(date +%Y%m%d%H%M").tar.gz $(DISTRO_FILES)

```

This code is used in chunk 208.

Unit test and target

```
210 <Makefile: unit tests and targets 210>≡
    pycalcaltests.py: pycalcal.py
        $(NOTANGLE) -filter btdefn -R$.py - $(CPIF) $.py

.PHONY : test
test: pycalcaltests.py $(NW_MAIN:.nw=.py) testdata ut
    python pycalcaltests.py 2>&1 | tee testResult.txt

.PHONY : singletest
singletest:
    for t in $$$(cat Makefile | grep 'UnitTest.py:' | \
        cut -f1 -d':.' | grep -v appendix); \
    do \
        $(MAKE) $$t; \
    done

# create all unit test files
.PHONY : ut
ut: pycalcaltests.py
    $(MAKE) $(UNIT_TEST_FILES)

<Makefile: basics unit tests and target 211>
<Makefile: egyptian and armenian unit tests and target 212>
<Makefile: gregorian unit tests and target 213>
<Makefile: iso unit tests and target 214>
<Makefile: julian unit tests and target 215>
<Makefile: coptic and ethiopic unit tests and target 216>
<Makefile: ecclesiastical unit tests and target 217>
<Makefile: islamic unit tests and target 218>
<Makefile: hebrew unit tests and target 219>
<Makefile: mayan unit tests and target 220>
<Makefile: old hindu unit tests and target 221>
<Makefile: balinese unit tests and target 222>
<Makefile: time and astronomy unit tests and target 223>
<Makefile: persian unit tests and target 224>
<Makefile: bahai unit tests and target 225>
<Makefile: french unit tests and target 226>
<Makefile: chinese unit tests and target 227>
<Makefile: modern hindu unit tests and target 228>
<Makefile: tibetan unit tests and target 229>
<Makefile: astronomical lunar unit tests and target 230>
<Makefile: appendix c unit tests and target 231>
<Makefile: coverage target 187>

trasformLatexDates2Cvs:
    notangle -RtrasformLatexDates2Cvs $(NW_MAIN) $(CPIF) trasformLatexDates2Cvs
    -chmod ug+x trasformLatexDates2Cvs
```

This code is used in chunk 208.

211 *<Makefile: basics unit tests and target 211>≡*
`basicCodeUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : basicCodeUnitTest`
`basicCodeUnitTest: basicCodeUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

212 *<Makefile: egyptian and armenian unit tests and target 212>≡*
`egyptianAndArmenianCalendarsUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : egyptianAndArmenianCalendarsUnitTest`
`egyptianAndArmenianCalendarsUnitTest: egyptianAndArmenianCalendarsUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

213 *<Makefile: gregorian unit tests and target 213>≡*
`gregorianCalendarUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : gregorianCalendarUnitTest`
`gregorianCalendarUnitTest: gregorianCalendarUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

214 *<Makefile: iso unit tests and target 214>≡*

`isoCalendarUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : isoCalendarUnitTest`
`isoCalendarUnitTest: isoCalendarUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

215 *<Makefile: julian unit tests and target 215>≡*
`julianCalendarUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : julianCalendarUnitTest`
`julianCalendarUnitTest: julianCalendarUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

216 *<Makefile: coptic and ethiopic unit tests and target 216>≡*
`copticAndEthiopicCalendarsUnitTest.py: appendixCUnitTest.py`
`$(NOTANGLE) -filter btdefn -R$.py - > $.py`

`.PHONY : copticAndEthiopicCalendarsUnitTest`
`copticAndEthiopicCalendarsUnitTest: copticAndEthiopicCalendarsUnitTest.py`
`python $@.py 2>&1 | tee $@_result.txt`

This code is used in chunk 210.

217 *<Makefile: ecclesiastical unit tests and target 217>≡*
 ecclesiasticalCalendarsUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : ecclesiasticalCalendarsUnitTest
 ecclesiasticalCalendarsUnitTest: ecclesiasticalCalendarsUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

218 *<Makefile: islamic unit tests and target 218>≡*
 islamicCalendarUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : islamicCalendarUnitTest
 islamicCalendarUnitTest: islamicCalendarUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

219 *<Makefile: hebrew unit tests and target 219>≡*
 hebrewCalendarUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : hebrewCalendarUnitTest
 hebrewCalendarUnitTest: hebrewCalendarUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

220 *<Makefile: mayan unit tests and target 220>≡*
 mayanCalendarsUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : mayanCalendarsUnitTest
 mayanCalendarsUnitTest: mayanCalendarsUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

221 *<Makefile: old hindu unit tests and target 221>≡*
 oldHinduCalendarsUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : oldHinduCalendarsUnitTest
 oldHinduCalendarsUnitTest: oldHinduCalendarsUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

222 *<Makefile: balinese unit tests and target 222>≡*
 balineseCalendarUnitTest.py: appendixCUnitTest.py
 \$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

 .PHONY : balineseCalendarUnitTest
 balineseCalendarUnitTest: balineseCalendarUnitTest.py
 python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

223 *<Makefile: time and astronomy unit tests and target 223>*≡
timeAndAstronomyUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : timeAndAstronomyUnitTest
timeAndAstronomyUnitTest: timeAndAstronomyUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

224 *<Makefile: persian unit tests and target 224>*≡
persianCalendarUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : persianCalendarUnitTest
persianCalendarUnitTest: persianCalendarUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

225 *<Makefile: bahai unit tests and target 225>*≡
bahaiCalendarUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : bahaiCalendarUnitTest
bahaiCalendarUnitTest: bahaiCalendarUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

226 *<Makefile: french unit tests and target 226>*≡
frenchRevolutionaryCalendarUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : frenchRevolutionaryCalendarUnitTest
frenchRevolutionaryCalendarUnitTest: frenchRevolutionaryCalendarUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

227 *<Makefile: chinese unit tests and target 227>*≡
chineseCalendarUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : chineseCalendarUnitTest
chineseCalendarUnitTest: chineseCalendarUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

228 *<Makefile: modern hindu unit tests and target 228>*≡
modernHinduCalendarsUnitTest.py: appendixCUnitTest.py
\$(NOTANGLE) -filter btdefn -R\$*.py - > \$*.py

.PHONY : modernHinduCalendarsUnitTest
modernHinduCalendarsUnitTest: modernHinduCalendarsUnitTest.py
python \$@.py 2>&1 | tee \$@_result.txt

This code is used in chunk 210.

```

229 <Makefile: tibetan unit tests and target 229>≡
    tibetanCalendarUnitTest.py: appendixCUnitTest.py
        $(NOTANGLE) -filter btdefn -R$*.py - > $*.py

    .PHONY : tibetanCalendarUnitTest
    tibetanCalendarUnitTest: tibetanCalendarUnitTest.py
        python $@.py 2>&1 | tee $@_result.txt

```

This code is used in chunk 210.

```

230 <Makefile: astronomical lunar unit tests and target 230>≡
    astronomicalLunarCalendarsUnitTest.py: appendixCUnitTest.py
        $(NOTANGLE) -filter btdefn -R$*.py - > $*.py

    .PHONY : astronomicalLunarCalendarsUnitTest
    astronomicalLunarCalendarsUnitTest: astronomicalLunarCalendarsUnitTest.py
        python $@.py 2>&1 | tee $@_result.txt

```

This code is used in chunk 210.

```

231 <Makefile: appendix c unit tests and target 231>≡
    appendixCUnitTest.py: $(NW_MAIN:.nw=.py)
        $(NOTANGLE) -filter btdefn -R$*.py - > $*.py

    .PHONY : appendixCUnitTest
    appendixCUnitTest: appendixCUnitTest.py
        python $@.py 2>&1 | tee $@_result.txt

    .PHONY : testdata
    testdata: trasformLatexDates2Cvs
        cat dates1.tex | ./trasformLatexDates2Cvs > dates1.csv
        cat dates2.tex | ./trasformLatexDates2Cvs > dates2.csv
        cat dates3.tex | ./trasformLatexDates2Cvs > dates3.csv
        cat dates4.tex | ./trasformLatexDates2Cvs > dates4.csv
        cat dates5.tex | ./trasformLatexDates2Cvs > dates5.csv

```

This code is used in chunk 210.

```

232 <Makefile: tools 232>≡
    <Makefile: premarkup 195>
    <Makefile: prefix 196>
    <Makefile: excerpt 198>
    <Makefile: blankpremarkup 201>
    <Makefile: blankexcerpt 202>

    sconstruct:
        notangle -RSConstruct $(NW_MAIN) $(CPIF) SConstruct

```

This code is used in chunk 208.


```

233 <Makefile: clean 233>≡
    .PHONY : clean clobber superclean
    clobber: clean

    clean:
        rm -fR pycalcal.tex $(NW_MAIN:.nw=.py) pycalcaltests.py *.dvi *.aux *.log \
            *.blg *.toc *.bbl *~ *.pyc *.defs *.nwi premarkup prefix \
            excerpt dates?.csv blankexcerpt blankpremarkup \
            trasformLatexDates2Cvs pycalcal.pdf extractcc3signatures \
            extractcalcalsignatures pycalcal.ind pycalcal.out pycalcal.ilg \
            pycalcal.idx *UnitTest.py *UnitTest_result.txt \
            html/ calendrica/ pycalcal*.gz figure.mpx figure.0 figure-*.pdf

    superclean: clean
        for f in $$ (hg status | grep -e '^?' | sed -e 's/^? //'g); \
        do \
            rm -i $$f; \
        done

```

This code is used in chunk 208.

4.4 SConstruct It

Here is my experiment with SCons

```

234 <SConstruct 234>≡
    # <generated code warning 1>
    <scons preamble 235>
    <scons notangle 236>
    <scons noweave 237>
    <scons noindex 238>
    <scons pycalcal preamble 239>
    <scons pycalcal tools 240>
    <scons pycalcal targets 241>

```

Root chunk (not used in this document).

```

235 <scons preamble 235>≡
    import os

    dbg = Environment()
    env = Environment(ENV = os.environ)
    env.PrependENVPath('PATH', '..')
    EnsurePythonVersion(2,5)

    spitsrc = 'cat $SOURCE'
    pre     = 'premarkup'
    pipe    = ' | '

    # noweb install dir
    noweb_home = '/usr/local/noweb/bin/'

```

This code is used in chunk 234.

```

236 <scons notangle 236>≡
    # notangle consts
    ntfile    = noweb_home + 'notangle'
    ntflags   = '-filter btdefn -R$TARGET'
    ntcmd     = ntfile + ' ' + ntflags + ' -> $TARGET'

    prent     = spitsrc + pipe + \
                pre      + pipe + \
                ntcmd

    nt        = spitsrc + pipe + \
                ntcmd

    env.Append(BUILDERS={'NoTangle': Builder(action = nt)})
    dbg.Append(BUILDERS={'NoTangle': Builder(action = prent)})

```

This code is used in chunk 234.

```

237 <scons noweave 237>≡
    # noweave consts
    nwfile    = noweb_home + 'noweave'
    nwflags   = ' -n -delay -filter btdefn -index'
    nwcmd     = nwfile + ' ' + nwflags + ' -> $TARGET'

    prenw     = spitsrc + pipe + \
                pre      + pipe + \
                nwcmd

    nw        = spitsrc + pipe + \
                nwcmd

    env.Append(BUILDERS={'NoWeave': Builder(action = nw)})
    dbg.Append(BUILDERS={'NoWeave': Builder(action = prenw)})

```

This code is used in chunk 234.

```

238 <scons noindex 238>≡
    # noindex
    nifile    = noweb_home + 'noindex'
    niflags   = "
    nicmd     = nifile + ' ' + niflags + ' $SOURCE'

    ni        = nicmd

    env.Append(BUILDERS={'NoIndex': Builder(action = ni)})
    dbg.Append(BUILDERS={'NoIndex': Builder(action = ni)})

```

This code is used in chunk 234.

```

239 <scons pycalcal preamble 239>≡
#####
# pycalcal specific
#####
builder=env
unit_test_files="""basicCodeUnitTest.py
                    egyptianAndArmenianCalendarsUnitTest.py
                    gregorianCalendarUnitTest.py
                    isoCalendarUnitTest.py
                    julianCalendarUnitTest.py
                    copticAndEthiopicCalendarsUnitTest.py
                    ecclesiasticalCalendarsUnitTest.py
                    islamicCalendarUnitTest.py
                    hebrewCalendarUnitTest.py
                    mayanCalendarsUnitTest.py
                    oldHinduCalendarsUnitTest.py
                    balineseCalendarUnitTest.py
                    timeAndAstronomyUnitTest.py
                    persianCalendarUnitTest.py
                    bahaiCalendarUnitTest.py
                    frenchRevolutionaryCalendarUnitTest.py
                    chineseCalendarUnitTest.py
                    modernHinduCalendarsUnitTest.py
                    tibetanCalendarUnitTest.py
                    astronomicalLunarCalendarsUnitTest.py"""

```

This code is used in chunk 234.

```

240 <scons pycalcal tools 240>≡
# tools
prefix = builder.NoTangle('prefix', 'pycalcal.nw')
excerpt = builder.NoTangle('excerpt', 'pycalcal.nw')
premarkup = builder.NoTangle('premarkup', 'pycalcal.nw')
Requires(premarkup, [excerpt, prefix])

```

This code is used in chunk 234.

```

241 <scons pycalcal targets 241>≡
pycalcal_code = dbg.NoTangle('pycalcal.py', 'pycalcal.nw',)
# Requires(pycalcal_code, premarkup)
Clean(pycalcal_code, Split('pycalcal.pyo pycalcal.pyc'))

pycalcal_tests = builder.NoTangle('pycalcaltests.py', 'pycalcal.nw',)
Requires(pycalcal_tests, [premarkup, pycalcal_code])
Clean(pycalcal_tests, Split('pycalcaltests.pyo pycalcaltestst.pyc'))

pycalcal_latex = builder.NoWeave('pycalcal.tex', 'pycalcal.nw')
Requires(pycalcal_latex, premarkup)
Clean(pycalcal_latex, 'pycalcal.out')

pycalcal_noindex = builder.NoIndex('pycalcal.nwi', 'pycalcal.tex')
Depends(pycalcal_noindex, pycalcal_latex)
pycalcal_pdf = builder.PDF(target = 'pycalcal.pdf',
                           source = 'pycalcal.tex')

Depends(pycalcal_pdf, Split('pycalcal.tex pycalcal.bib'))
Default(pycalcal_pdf, pycalcal_code, pycalcal_tests)

```

This code is used in chunk 234.

4.5 Floating-point nuances

A snippet of COMMON LISP code from Prof. Reingold.

```
242 <testSunset.cl 242>≡
    #!/usr/bin/env clisp
    ;; -*- Mode: Lisp -*-

    (load "calendrica-3.0.cl")
    ;;(in-package "CC3")
    ;;(use-package "CC3")
    (defun format-time (p)
      (if (equal p CC3:bogus)
          "\\multicolumn{4}{c}{\\fun{bogus}}"
          ;;; add 1/2 second...
          (let ((time (CC3:clock-from-moment (+ (/ 0.5 24 60 60) p))))
            (list (mod p 1)
                  (format nil "~2,'OD" (first time))
                  (format nil "~2,'OD" (second time))
                  ;;; ...and truncate
                  (format nil "~2,'OD" (floor (third time)))))))

    (defvar *moment*
      (and (= 1 (length *args*))
           (ignore-errors
            (with-input-from-string (strm (first *args*))
              (read strm)))))

    (if (integerp *moment*)
        (format-time (CC3:sunrise *moment* CC3:paris))
        (format t "~&error on the command line-&"))
```

Root chunk (not used in this document).

Uses mod 15 and sunrise 92.

4.6 Small differences or errors?

This is a python script to output values from “Calendrical Calculations” and the ones from my library.

```
243 <printObservationalHebrewFromFixed.py 243>≡
    from pycalcal import *
    import csv

    rd = []
    ho = []
    reader = csv.reader(open("dates3.csv", "rU"),
                        delimiter=',',
                        quoting=csv.QUOTE_NONE)

    for row in reader:
        rd.append(int(row[0]))
        ho.append(hebrew_date(int(row[4]), int(row[5]), int(row[6])))

    for i in range(len(rd)):
        # observational
        print 'R.D.= %d; oh_from_fxd=%s; fxd_from_oh=%d; book=%s' % (rd[i],
                            observational_hebrew_from_fixed(rd[i]),
                            fixed_from_observational_hebrew(ho[i]),
                            ho[i])
```

Root chunk (not used in this document).

Uses `fixed_from_observational_hebrew` 132, `hebrew_date` 80, `observational_hebrew_from_fixed` 132, and `rd` 37.

4.7 Floating-point nuances

While investigating the differences from my results and COMMON LISP ones, I got a nice email exchange with Prof. Reingold. I was pointing out that COMMON LISP on my machine was not returning the same numbers as in the Appendix C’ last table. He ran it on SPARC and on Intel x86 ... and bang!

On the PYTHON side, my tests fail for “Dawn in Paris” if I force 6 digits of precision, otherwise they all pass but one (on RD 601716) when 4 digits are taken: this last paragraph is not true. I found this was due to the wrong implementation of angle: division was with integers and not floating point numbers!

4.8 Chasing bugs

Observational Hebrew conversion tests fail for (almost all) Appendix C dates. I try to find what's wrong in `observational_hebrew_from_fixed`: `gregorian_year_from_fixed` seems to be ok; `observational_hebrew_new_year` is not; `gregorian_new_year` is ok.

```
[1] (load "calendrica-3.0.cl")
[4]> (CC3:observational-hebrew-new-year (CC3:gregorian-year-from-fixed -214193))
-214319
[5]> (CC3:observational-hebrew-new-year (CC3:gregorian-year-from-fixed -61387))
-61647
[6]> (CC3:gregorian-year-from-fixed -214193)
-586
[7]> (CC3:gregorian-year-from-fixed -61387)
-168
[8]> (CC3:gregorian-year-from-fixed 25469)
70
[9]> (CC3:gregorian-year-from-fixed 613424)
1680
[10]> (CC3:gregorian-year-from-fixed 744313)
2038
[11]> (CC3:solar-longitude-after CC3:spring (CC3:gregorian-new-year 764652))
2.7928327878524209297L8
```

and python:

```
>>> p.solar_longitude_after(p.SPRING, p.gregorian_new_year(764652))
p.solar_longitude_after(p.SPRING, p.gregorian_new_year(764652))
mpf('279283278.7852416')
```

Bibliography

- [1] Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, third edition, 2008. Common Lisp code available at http://www.cup.cam.ac.uk/resources/0521702380/5106_calendrica-3.0.cl.zip.
- [2] Google app engine. <http://code.google.com/appengine/>.
- [3] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, pub-SUCSLI:adr, 1992.
- [4] mpmath. <http://code.google.com/p/mpmath/>.
- [5] Jonathan Normand and Jérôme Berthier. Institut de mécanique céleste – observatoire de paris. éphémérides astronomiques. <http://www.imcce.fr/page.php?nav=fr/ephemerides/phenomenes/rts/>. dat service on ephemerides.
- [6] Python. <http://www.python.org/>.
- [7] Norman Ramsey. Noweb home page. <http://www.cs.tufts.edu/~nr/noweb/>.
- [8] Norman Ramsey. Literate programming simplified. *IEEE Softw.*, 11(5):97–105, 1994.
- [9] Gareth Rees and Ned Batchelder. Coverage.py – a tool for measuring code coverage of python programs. <http://nedbatchelder.com/code/coverage/>.
- [10] Scons: a software construction tool. <http://www.scons.org/>.
- [11] Test-driven development. http://en.wikipedia.org/wiki/Test-driven_development.
- [12] Data services - astronomical applications. <http://www.usno.navy.mil/USNO/astronomical-applications/data-services/rs-one-day-world>. Web based data services from US Naval Observatory. Sunset/rise, twilight, phases of the Moon.
- [13] Moshe Zadka and Guido van Rossum. Pep 238 – changing the division operator. <http://www.python.org/dev/peps/pep-0238/>.

Appendix A

Chunks

A.1 Chunks Index

[⟨ * 2 ⟩](#) [2](#)
[⟨ appendix c unit test 138 ⟩](#) [137](#), [138](#)
[⟨ appendixCTable1Tests 139 ⟩](#) [138](#), [139](#)
[⟨ appendixCTable2Tests 155 ⟩](#) [138](#), [155](#)
[⟨ appendixCTable3Tests 164 ⟩](#) [138](#), [164](#)
[⟨ appendixCTable4Tests 175 ⟩](#) [138](#), [175](#)
[⟨ appendixCTable5Tests 184 ⟩](#) [138](#), [184](#)
[⟨ appendixCUnitTest.py 137 ⟩](#) [137](#)
[⟨ app.yaml 192 ⟩](#) [192](#)
[⟨ astronomical algorithms tests 93 ⟩](#) [93](#), [94](#)
[⟨ astronomical lunar calendars 95 ⟩](#) [3](#), [95](#), [97](#), [99](#), [101](#), [103](#), [105](#), [107](#), [109](#), [110](#), [132](#)
[⟨ astronomical lunar calendars unit test 135 ⟩](#) [4](#), [134](#), [135](#)
[⟨ astronomical lunar tests 96 ⟩](#) [96](#), [98](#), [100](#), [102](#), [104](#), [106](#), [108](#), [135](#)
[⟨ astronomicalLunarCalendarsUnitTest.py 134 ⟩](#) [134](#)
[⟨ bahai calendar 115 ⟩](#) [3](#), [115](#)
[⟨ bahai calendar unit test 116 ⟩](#) [4](#), [116](#), [117](#), [160](#)
[⟨ bahaiCalendarUnitTest.py 117 ⟩](#) [117](#)
[⟨ balinese calendar 89 ⟩](#) [3](#), [89](#)
[⟨ balinese calendar unit test 90 ⟩](#) [4](#), [90](#), [91](#), [169](#)
[⟨ balineseCalendarUnitTest.py 91 ⟩](#) [91](#)
[⟨ basic code 13 ⟩](#) [3](#), [13](#), [14](#), [15](#), [16](#), [19](#), [22](#), [25](#), [28](#), [31](#), [34](#), [37](#), [38](#), [39](#), [40](#), [43](#), [46](#), [47](#), [48](#)
[⟨ basic code tests 17 ⟩](#) [17](#), [20](#), [23](#), [26](#), [29](#), [32](#), [35](#), [41](#), [44](#), [50](#)
[⟨ basic code unit test 50 ⟩](#) [4](#), [49](#), [50](#), [140](#)
[⟨ basicCodeUnitTest.py 49 ⟩](#) [49](#)
[⟨ blankexcerpt 199 ⟩](#) [199](#)
[⟨ blankpremarkup 200 ⟩](#) [200](#)
[⟨ calendrica.py 191 ⟩](#) [191](#)
[⟨ chinese calendar 123 ⟩](#) [3](#), [123](#)
[⟨ chinese calendar unit test 124 ⟩](#) [4](#), [124](#), [125](#), [176](#)
[⟨ chineseCalendarUnitTest.py 125 ⟩](#) [125](#)
[⟨ coda 133 ⟩](#) [3](#), [133](#)
[⟨ coptic and ethiopic calendars 71 ⟩](#) [3](#), [71](#)
[⟨ coptic and ethiopic calendars unit test 72 ⟩](#) [4](#), [72](#), [73](#), [153](#), [156](#)
[⟨ copticAndEthiopicCalendarsUnitTest.py 73 ⟩](#) [73](#)
[⟨ copyright Dershowitz and Reingold 8 ⟩](#) [8](#)
[⟨ ecclesiastical calendars 74 ⟩](#) [3](#), [74](#)
[⟨ ecclesiastical calendars unit test 75 ⟩](#) [4](#), [75](#), [76](#), [167](#)
[⟨ ecclesiasticalCalendarsUnitTest.py 76 ⟩](#) [76](#)
[⟨ egyptian and armenian calendars 65 ⟩](#) [3](#), [65](#)

<egyptian and armenian calendars unit test 66> [4](#), [66](#), [67](#), [152](#)
 <egyptianAndArmenianCalendarsUnitTest.py 67> [67](#)
 <excerpt 197> [197](#)
 <execute tests 5> [4](#), [5](#), [49](#), [58](#), [64](#), [67](#), [70](#), [73](#), [76](#), [79](#), [82](#), [85](#), [88](#), [91](#), [111](#), [114](#), [117](#), [122](#),
[125](#), [128](#), [131](#), [134](#), [137](#)
 <extractcalcsignatures 189> [189](#)
 <extractcc3signatures 188> [188](#)
 <french revolutionary calendar 118> [3](#), [118](#), [120](#)
 <french revolutionary calendar unit test 119> [4](#), [119](#), [121](#), [122](#), [173](#)
 <frenchRevolutionaryCalendarUnitTest.py 122> [122](#)
 <generated code warning 1> [1](#), [4](#), [9](#), [49](#), [58](#), [64](#), [67](#), [70](#), [73](#), [76](#), [79](#), [82](#), [85](#), [88](#), [91](#), [111](#), [114](#),
[117](#), [122](#), [125](#), [128](#), [131](#), [134](#), [137](#), [188](#), [191](#), [193](#), [194](#), [197](#), [203](#), [234](#)
 <global import statements 11> [3](#), [11](#), [12](#)
 <gregorian calendar 51> [3](#), [51](#)
 <gregorian calendar unit test 57> [4](#), [57](#), [58](#), [144](#)
 <gregorian conversion functions 55> [51](#), [55](#)
 <gregorian date and epoch 52> [51](#), [52](#)
 <gregorian leap year function 54> [51](#), [54](#)
 <gregorian months 53> [51](#), [53](#)
 <gregorian year start and end 56> [51](#), [56](#)
 <gregorianCalendarUnitTest.py 58> [58](#)
 <hebrew calendar 80> [3](#), [80](#)
 <hebrew calendar unit test 81> [4](#), [81](#), [82](#), [165](#)
 <hebrewCalendarUnitTest.py 82> [82](#)
 <import for testing 6> [6](#), [49](#), [58](#), [64](#), [67](#), [70](#), [73](#), [76](#), [79](#), [82](#), [85](#), [88](#), [91](#), [111](#), [114](#), [117](#), [122](#),
[125](#), [128](#), [131](#), [134](#), [137](#)
 <islamic calendar 77> [3](#), [77](#)
 <islamic calendar unit test 78> [4](#), [78](#), [79](#), [158](#)
 <islamicCalendarUnitTest.py 79> [79](#)
 <iso calendar 68> [3](#), [68](#)
 <iso calendar unit test 69> [4](#), [69](#), [70](#), [146](#)
 <isoCalendarUnitTest.py 70> [70](#)
 <julian calendar 59> [3](#), [59](#), [61](#), [63](#)
 <julian calendar unit test 60> [4](#), [60](#), [62](#), [64](#), [142](#), [149](#)
 <julianCalendarUnitTest.py 64> [64](#)
 <LICENSE 7> [4](#), [7](#), [9](#)
 <Makefile 203> [203](#)
 <Makefile: appendix c unit tests and target 231> [210](#), [231](#)
 <Makefile: astronomical lunar unit tests and target 230> [210](#), [230](#)
 <Makefile: bahai unit tests and target 225> [210](#), [225](#)
 <Makefile: balinese unit tests and target 222> [210](#), [222](#)
 <Makefile: basics unit tests and target 211> [210](#), [211](#)
 <Makefile: blankexcerpt 202> [202](#), [232](#)
 <Makefile: blankpremarkup 201> [201](#), [232](#)
 <Makefile: chinese unit tests and target 227> [210](#), [227](#)
 <Makefile: clean 233> [208](#), [233](#)
 <Makefile: commands 206> [203](#), [206](#)
 <Makefile: coptic and ethiopic unit tests and target 216> [210](#), [216](#)
 <Makefile: coverage target 187> [187](#), [210](#)
 <Makefile: cross checks 190> [190](#), [208](#)
 <Makefile: distro 209> [208](#), [209](#)
 <Makefile: ecclesiastical unit tests and target 217> [210](#), [217](#)
 <Makefile: egyptian and armenian unit tests and target 212> [210](#), [212](#)
 <Makefile: excerpt 198> [198](#), [232](#)
 <Makefile: files declarations 205> [203](#), [205](#)
 <Makefile: french unit tests and target 226> [210](#), [226](#)

<Makefile: gregorian unit tests and target 213> [210](#), [213](#)
 <Makefile: heading comment 204> [203](#), [204](#)
 <Makefile: hebrew unit tests and target 219> [210](#), [219](#)
 <Makefile: islamic unit tests and target 218> [210](#), [218](#)
 <Makefile: iso unit tests and target 214> [210](#), [214](#)
 <Makefile: julian unit tests and target 215> [210](#), [215](#)
 <Makefile: mayan unit tests and target 220> [210](#), [220](#)
 <Makefile: modern hindu unit tests and target 228> [210](#), [228](#)
 <Makefile: old hindu unit tests and target 221> [210](#), [221](#)
 <Makefile: persian unit tests and target 224> [210](#), [224](#)
 <Makefile: prefix 196> [196](#), [232](#)
 <Makefile: premarkup 195> [195](#), [232](#)
 <Makefile: suffixes 207> [203](#), [207](#)
 <Makefile: targets 208> [203](#), [208](#)
 <Makefile: tibetan unit tests and target 229> [210](#), [229](#)
 <Makefile: time and astronomy unit tests and target 223> [210](#), [223](#)
 <Makefile: tools 232> [208](#), [232](#)
 <Makefile: unit tests and targets 210> [208](#), [210](#)
 <mayan calendars 83> [3](#), [83](#)
 <mayan calendars unit test 84> [4](#), [84](#), [85](#), [162](#)
 <mayanCalendarsUnitTest.py 85> [85](#)
 <modern hindu calendars 126> [3](#), [126](#)
 <modern hindu calendars unit test 127> [4](#), [127](#), [128](#), [180](#)
 <modernHinduCalendarsUnitTest.py 128> [128](#)
 <old hindu calendars 86> [3](#), [86](#)
 <old hindu calendars unit test 87> [4](#), [87](#), [88](#), [178](#)
 <oldHinduCalendarsUnitTest.py 88> [88](#)
 <persian calendar 112> [3](#), [112](#)
 <persian calendar unit test 113> [4](#), [113](#), [114](#), [171](#)
 <persianCalendarUnitTest.py 114> [114](#)
 <prefix 194> [194](#)
 <premarkup 193> [193](#)
 <printObservationalHebrewFromFixed.py 243> [243](#)
 <project version 10> [10](#), [203](#)
 <pycalcal.py 3> [2](#), [3](#)
 <pycalcaltests.py 4> [4](#)
 <scons noindex 238> [234](#), [238](#)
 <scons notangle 236> [234](#), [236](#)
 <scons nowave 237> [234](#), [237](#)
 <scons preamble 235> [234](#), [235](#)
 <scons pycalcal preamble 239> [234](#), [239](#)
 <scons pycalcal targets 241> [234](#), [241](#)
 <scons pycalcal tools 240> [234](#), [240](#)
 <SConstruct 234> [234](#)
 <test armenian appendix c 151> [139](#), [151](#), [152](#)
 <test astronomy appendix c 186> [184](#), [185](#), [186](#)
 <test bahai appendix c 161> [155](#), [160](#), [161](#)
 <test balinese appendix c 170> [164](#), [169](#), [170](#)
 <test basic appendix c 141> [139](#), [140](#), [141](#)
 <test binary search 27> [26](#), [27](#)
 <test chinese appendix c 177> [175](#), [176](#), [177](#)
 <test clock from moment 42> [41](#), [42](#)
 <test coptic appendix c 154> [139](#), [153](#), [154](#)
 <test easter appendix c 168> [164](#), [167](#), [168](#)
 <test egyptian appendix c 150> [139](#), [150](#), [152](#)
 <test ethiopic appendix c 157> [155](#), [156](#), [157](#)

<test final 21> [20](#), [21](#)
 <test french appendix c 174> [164](#), [173](#), [174](#)
 <test gregorian appendix c 145> [139](#), [144](#), [145](#)
 <test hebrew appendix c 166> [164](#), [165](#), [166](#)
 <test invert angular 30> [29](#), [30](#)
 <test islamic appendix c 159> [155](#), [158](#), [159](#)
 <test iso appendix c 147> [139](#), [146](#), [147](#)
 <test julian appendix c 148> [139](#), [148](#), [149](#)
 <test julian day appendix c 143> [142](#), [143](#)
 <test mayan appendix c 163> [155](#), [162](#), [163](#)
 <test modern hindu appendix c 181> [175](#), [180](#), [181](#)
 <test next 18> [17](#), [18](#)
 <test old hindu appendix c 179> [175](#), [178](#), [179](#)
 <test persian appendix c 172> [164](#), [171](#), [172](#)
 <test poly 36> [35](#), [36](#)
 <test sigma 33> [32](#), [33](#)
 <test summa 24> [23](#), [24](#)
 <test tibetan appendix c 183> [175](#), [182](#), [183](#)
 <test time from clock 45> [44](#), [45](#)
 <testa 9> [3](#), [9](#)
 <testSunset.cl 242> [242](#)
 <tibetan calendar 129> [3](#), [129](#)
 <tibetan calendar unit test 130> [4](#), [130](#), [131](#), [182](#)
 <tibetanCalendarUnitTest.py 131> [131](#)
 <time and astronomy 92> [3](#), [92](#)
 <time and astronomy unit test 94> [4](#), [94](#), [111](#), [185](#)
 <timeAndAstronomyUnitTest.py 111> [111](#)
 <trasformLatexDates2Cvs 136> [136](#)

A.2 Chunks Identifiers

aberration: [97](#), [99](#)
 ADAR: [80](#)
 ADARII: [80](#)
 advent: [56](#)
 alt_fixed_from_gregorian: [56](#), [145](#)
 alt_gregorian_from_fixed: [56](#), [145](#)
 alt_gregorian_year_from_fixed: [56](#), [145](#)
 alt_hindu_sunrise: [126](#)
 alt_lunar_node: [109](#)
 alt_orthodox_easter: [74](#), [168](#)
 altsumma: [22](#), [24](#)
 alt_tzom_tevet: [80](#)
 amod: [15](#), [56](#), [61](#), [68](#), [83](#), [86](#), [89](#), [123](#), [126](#), [129](#)
 angle: [30](#), [46](#), [92](#), [95](#), [97](#), [109](#), [110](#), [118](#), [123](#), [126](#), [132](#), [186](#)
 angle_from_degrees: [46](#)
 apparent_from_local: [92](#), [132](#)
 AppendixCTable1TestCase: [139](#)
 AppendixCTable2TestCase: [155](#)
 AppendixCTable3TestCase: [164](#)
 AppendixCTable4TestCase: [175](#)
 AppendixCTable5TestCase: [184](#)
 AppendixCTable1TestCaseBase: [4](#), [49](#), [58](#), [64](#), [67](#), [70](#), [73](#), [139](#), [140](#), [142](#), [144](#), [146](#), [149](#),
[152](#), [153](#)
 AppendixCTable2TestCaseBase: [4](#), [73](#), [79](#), [85](#), [117](#), [155](#), [156](#), [158](#), [160](#), [162](#)

AppendixCTable3TestCaseBase: [4](#), [76](#), [82](#), [91](#), [114](#), [122](#), [164](#), [165](#), [167](#), [169](#), [171](#), [173](#)
 AppendixCTable4TestCaseBase: [4](#), [88](#), [125](#), [128](#), [131](#), [175](#), [176](#), [178](#), [180](#), [182](#)
 AppendixCTable5TestCaseBase: [4](#), [111](#), [134](#), [184](#), [185](#)
 approx_moment_of_depression: [92](#)
 APRIL: [53](#), [74](#), [98](#), [123](#)
 arccos_degrees: [92](#), [132](#)
 arcsin_degrees: [92](#), [110](#)
 arctan_degrees: [92](#), [92](#), [95](#), [99](#)
 arithmetic_french_from_fixed: [120](#), [174](#)
 arithmetic_persian_from_fixed: [112](#), [172](#)
 arithmetic_persian_year_from_fixed: [112](#)
 ArmeniamAppendixCTestCase: [152](#)
 armenian_date: [65](#), [66](#), [139](#)
 ARMENIAN_EPOCH: [65](#)
 armenian_from_fixed: [65](#), [66](#), [151](#)
 ArmenianSmokeTestCase: [66](#)
 ARYA_JOVIAN_PERIOD: [86](#)
 ARYA_LUNAR_DAY: [86](#)
 ARYA_LUNAR_MONTH: [86](#)
 ARYA_SOLAR_MONTH: [86](#)
 ARYA_SOLAR_YEAR: [86](#)
 asr: [95](#)
 astro_hindu_calendar_year: [126](#)
 astro_hindu_lunar_from_fixed: [126](#), [181](#)
 astro_hindu_solar_from_fixed: [126](#), [181](#)
 astro_hindu_sunset: [126](#)
 astro_lunar_day_from_moment: [126](#)
 AstronomicalAlgorithmsTestCase: [94](#)
 astronomical_easter: [132](#), [168](#)
 AstronomicalLunarCalendarsTestCase: [135](#)
 AstronomyAppendixCTestCase: [185](#)
 auc_year_from_julian_year: [61](#)
 AUGUST: [53](#), [71](#), [83](#), [123](#)
 AUTUMN: [99](#), [120](#), [186](#)
 AV: [80](#)
 ayanamsha: [126](#)
 AYYAM_I_HA: [115](#)
 AZTEC_CORRELATION: [83](#)
 AztecSmokeTestCase: [84](#)
 AZTEC_TONALPOHUALLI_CORRELATION: [83](#)
 aztec_tonalpohualli_date: [83](#), [155](#)
 aztec_tonalpohualli_from_fixed: [83](#), [163](#)
 aztec_tonalpohualli_name: [83](#)
 aztec_tonalpohualli_number: [83](#)
 aztec_tonalpohualli_on_or_before: [83](#)
 aztec_tonalpohualli_ordinal: [83](#)
 AZTEC_XIHUITL_CORRELATION: [83](#)
 aztec_xihuitl_date: [83](#), [84](#), [155](#)
 aztec_xihuitl_day: [83](#)
 aztec_xihuitl_from_fixed: [83](#), [84](#), [163](#)
 aztec_xihuitl_month: [83](#)
 aztec_xihuitl_on_or_before: [83](#), [84](#)
 aztec_xihuitl_ordinal: [83](#)
 aztec_xihuitl_tonalpohualli_on_or_before: [83](#)
 aztec_xiuhmolpilli_designation: [83](#)
 aztec_xiuhmolpilli_from_fixed: [83](#)

aztec_xiuhmolpilli_name: [83](#)
 aztec_xiuhmolpilli_number: [83](#)
 BahaiAppendixCTestCase: [160](#)
 bahai_cycle: [115](#)
 bahai_date: [115](#), [155](#)
 bahai_day: [115](#)
 BAHAI_EPOCH: [115](#)
 bahai_from_fixed: [115](#), [161](#)
 bahai_major: [115](#)
 bahai_month: [115](#)
 bahai_new_year: [115](#)
 bahai_year: [115](#)
 bali_asatawara: [89](#)
 bali_asatawara_from_fixed: [89](#)
 bali_caturwara: [89](#)
 bali_caturwara_from_fixed: [89](#)
 bali_dasawara: [89](#)
 bali_dasawara_from_fixed: [89](#)
 bali_day_from_fixed: [89](#)
 bali_dwiwara: [89](#)
 bali_dwiwara_from_fixed: [89](#)
 BALI_EPOCH: [89](#)
 bali_luang: [89](#)
 bali_luang_from_fixed: [89](#)
 BalineseAppendixCTestCase: [169](#)
 balinese_date: [89](#), [90](#), [164](#)
 BalineseSmokeTestCase: [90](#)
 bali_on_or_before: [89](#), [90](#)
 bali_pancawara: [89](#)
 bali_pancawara_from_fixed: [89](#)
 bali_pawukon_from_fixed: [89](#), [90](#), [170](#)
 bali_sadwara: [89](#)
 bali_sadwara_from_fixed: [89](#)
 bali_sangawara: [89](#)
 bali_sangawara_from_fixed: [89](#)
 bali_saptawara: [89](#)
 bali_saptawara_from_fixed: [89](#)
 bali_triwara: [89](#)
 bali_triwara_from_fixed: [89](#)
 bali_week_from_fixed: [89](#)
 BasicAppendixCTestCase: [140](#)
 bce: [59](#), [61](#), [80](#), [86](#)
 binary_search: [25](#), [27](#), [28](#), [92](#), [126](#)
 birkath_ha_hama: [80](#), [81](#)
 BOGUS: [13](#), [56](#), [80](#), [83](#), [92](#), [95](#), [123](#), [186](#)
 BRUXELLES: [92](#)
 ce: [59](#), [71](#), [77](#), [112](#), [126](#)
 ceiling: [86](#), [86](#), [112](#), [123](#), [126](#), [129](#)
 chinese_age: [123](#)
 ChineseAppendixCTestCase: [176](#)
 chinese_branch: [123](#)
 chinese_cycle: [123](#)
 chinese_date: [123](#), [175](#)
 chinese_day: [123](#)
 chinese_day_name: [123](#), [177](#)
 CHINESE_DAY_NAME_EPOCH: [123](#)

chinese_day_name_on_or_before: [123](#)
 CHINESE_EPOCH: [123](#)
 chinese_from_fixed: [123](#), [177](#)
 chinese_leap: [123](#)
 chinese_location: [123](#)
 chinese_month: [123](#)
 chinese_month_name: [123](#)
 CHINESE_MONTH_NAME_EPOCH: [123](#)
 chinese_name: [123](#)
 chinese_name_difference: [123](#)
 chinese_new_moon_before: [123](#)
 chinese_new_moon_on_or_after: [123](#)
 chinese_new_year: [123](#)
 chinese_new_year_in_sui: [123](#)
 chinese_new_year_on_or_before: [123](#)
 chinese_sexagesimal_name: [123](#)
 chinese_solar_longitude_on_or_after: [123](#)
 chinese_stem: [123](#)
 chinese_winter_solstice_on_or_before: [123](#)
 chinese_year: [123](#)
 chinese_year_marriage_augury: [123](#)
 chinese_year_name: [123](#)
 christmas: [56](#)
 classical_passover_eve: [132](#)
 clock_from_moment: [40](#), [42](#), [96](#)
 CopticAppendixCTestCase: [153](#)
 coptic_christmas: [71](#)
 coptic_date: [71](#), [72](#), [139](#)
 COPTIC_EPOCH: [71](#)
 coptic_from_fixed: [71](#), [72](#), [80](#), [154](#)
 coptic_in_gregorian: [71](#), [80](#)
 CopticSmokeTestCase: [72](#)
 cosine_degrees: [92](#), [97](#), [99](#), [110](#), [132](#)
 current_major_solar_term: [123](#)
 current_minor_solar_term: [123](#)
 dawn: [92](#), [126](#), [184](#), [186](#)
 daylight_saving_end: [56](#)
 daylight_saving_start: [56](#)
 day_number: [56](#), [57](#)
 day_of_week_from_fixed: [38](#), [56](#), [68](#), [80](#), [141](#)
 days_in_hebrew_year: [80](#)
 days_remaining: [56](#)
 daytime_temporal_hour: [95](#)
 DECEMBER: [53](#), [56](#), [59](#), [63](#), [68](#), [129](#)
 declination: [92](#), [94](#), [95](#), [110](#)
 deg: [92](#), [94](#), [95](#), [97](#), [99](#), [101](#), [103](#), [105](#), [107](#), [109](#), [110](#), [112](#), [115](#), [123](#), [126](#), [132](#)
 degrees: [46](#), [92](#), [99](#), [101](#), [103](#), [105](#), [107](#), [109](#), [110](#), [123](#), [126](#)
 degrees_from_radians: [92](#)
 degrees_minutes_seconds: [46](#)
 direction: [92](#)
 diwali: [126](#)
 dragon_festival: [123](#)
 dusk: [92](#), [94](#), [95](#), [126](#), [132](#)
 dynamical_from_universal: [92](#), [95](#), [96](#)
 easter: [74](#), [164](#), [168](#)
 EasterAppendixCTestCase: [167](#)

eastern_orthodox_christmas: [63](#)
 EgyptianAppendixCTestCase: [152](#)
 egyptian_date: [65](#), [66](#), [139](#)
 EGYPTIAN_EPOCH: [65](#)
 egyptian_from_fixed: [65](#), [66](#), [150](#)
 EgyptianSmokeTestCase: [66](#)
 election_day: [56](#)
 elevation: [92](#)
 ELUL: [80](#)
 end: [47](#), [56](#), [89](#), [95](#), [118](#), [123](#), [126](#)
 ephemeris_correction: [95](#), [97](#)
 epiphany: [56](#)
 epiphany_it: [56](#)
 epoch: [37](#), [80](#), [86](#)
 equation_of_time: [92](#), [97](#)
 estimate_prior_solar_longitude: [99](#), [112](#), [115](#), [120](#), [123](#)
 EthiopicAppendixCTestCase: [156](#)
 ethiopic_date: [71](#), [72](#), [155](#)
 ETHIOPIC_EPOCH: [71](#)
 ethiopic_from_fixed: [71](#), [72](#), [157](#)
 EthiopicSmokeTestCase: [72](#)
 even: [89](#)
 EVENING: [92](#)
 feast_of_ridvan: [115](#)
 FEBRUARY: [53](#), [61](#), [86](#), [96](#), [123](#)
 final: [19](#), [21](#), [80](#), [109](#), [129](#)
 first_kday: [56](#)
 FIRST_QUARTER: [109](#), [132](#)
 fixed_from_arithmetic_french: [120](#), [174](#)
 fixed_from_arithmetic_persian: [112](#), [172](#)
 fixed_from_armenian: [65](#), [66](#), [151](#)
 fixed_from_astro_hindu_lunar: [126](#), [181](#)
 fixed_from_astro_hindu_solar: [126](#), [181](#)
 fixed_from_bahai: [115](#), [161](#)
 fixed_from_chinese: [123](#), [177](#)
 fixed_from_coptic: [71](#), [72](#), [154](#)
 fixed_from_egyptian: [65](#), [66](#), [150](#)
 fixed_from_ethiopic: [71](#), [72](#), [157](#)
 fixed_from_french: [120](#), [174](#)
 fixed_from_future_bahai: [115](#), [161](#)
 fixed_from_gregorian: [55](#), [56](#), [57](#), [59](#), [74](#), [92](#), [96](#), [98](#), [115](#), [118](#), [119](#), [123](#), [129](#), [145](#), [191](#)
 fixed_from_hebrew: [80](#), [81](#), [132](#), [166](#)
 fixed_from_hindu_fullmoon: [126](#)
 fixed_from_hindu_lunar: [126](#), [181](#)
 fixed_from_hindu_solar: [126](#), [181](#)
 fixed_from_islamic: [77](#), [78](#), [159](#)
 fixed_from_iso: [68](#), [69](#), [147](#)
 fixed_from_jd: [48](#), [65](#), [83](#), [89](#), [143](#)
 fixed_from_julian: [59](#), [60](#), [61](#), [63](#), [71](#), [74](#), [77](#), [80](#), [83](#), [86](#), [112](#), [148](#)
 fixed_from_mayan_long_count: [83](#), [84](#), [163](#)
 fixed_from_mjd: [48](#), [143](#)
 fixed_from_moment: [40](#), [92](#), [95](#), [126](#)
 fixed_from_observational_hebrew: [132](#), [166](#), [243](#)
 fixed_from_observational_islamic: [132](#), [159](#)
 fixed_from_old_hindu_lunar: [86](#), [87](#), [179](#)
 fixed_from_old_hindu_solar: [86](#), [87](#), [179](#)

fixed_from_persian: [112](#), [172](#)
 fixed_from_roman: [60](#), [61](#), [62](#), [148](#)
 fixed_from_tibetan: [129](#), [183](#)
 french_date: [118](#), [120](#), [164](#)
 FRENCH_EPOCH: [118](#), [120](#)
 french_from_fixed: [120](#), [174](#)
 french_new_year_on_or_before: [120](#)
 FrenchRevolutionaryAppendixCTestCase: [173](#)
 FRIDAY: [38](#), [56](#), [80](#), [139](#)
 FULL: [109](#), [132](#)
 future_bahai_from_fixed: [115](#), [161](#)
 future_bahai_new_year_on_or_before: [115](#)
 GREENWHICH: [92](#)
 GregorianAppendixCTestCase: [144](#)
 GregorianCalendarSmokeTestCase: [57](#)
 gregorian_date: [52](#), [56](#), [57](#), [59](#), [68](#), [74](#), [92](#), [96](#), [97](#), [98](#), [115](#), [118](#), [119](#), [123](#), [129](#), [139](#),
[164](#), [191](#)
 gregorian_date_difference: [56](#), [97](#)
 GREGORIAN_EPOCH: [52](#), [55](#), [56](#), [74](#)
 gregorian_from_fixed: [56](#), [57](#), [145](#), [168](#)
 gregorian_new_year: [56](#), [63](#), [68](#), [71](#), [77](#), [80](#), [97](#), [123](#), [126](#), [132](#)
 gregorian_year_end: [56](#), [68](#), [129](#)
 gregorian_year_from_fixed: [55](#), [56](#), [68](#), [80](#), [97](#), [112](#), [115](#), [123](#), [132](#), [145](#), [168](#)
 gregorian_year_range: [56](#), [63](#), [71](#), [77](#), [80](#), [89](#), [126](#), [129](#)
 HAIFA: [115](#)
 HebrewAppendixCTestCase: [165](#)
 hebrew_birthday: [80](#)
 hebrew_birthday_in_gregorian: [80](#)
 hebrew_calendar_elapsed_days: [80](#)
 hebrew_date: [80](#), [81](#), [132](#), [164](#), [243](#)
 HEBREW_EPOCH: [80](#)
 hebrew_from_fixed: [80](#), [81](#), [132](#), [166](#)
 HebrewHolidaysTestCase: [81](#)
 hebrew_in_gregorian: [80](#)
 hebrew_new_year: [80](#)
 HebrewSmokeTestCase: [81](#)
 hebrew_year_length_correction: [80](#)
 HINDU_ANOMALISTIC_MONTH: [126](#)
 HINDU_ANOMALISTIC_YEAR: [126](#)
 hindu_arcsin: [126](#)
 hindu_ascensional_difference: [126](#)
 hindu_calendar_year: [126](#)
 HINDU_CREATION: [126](#)
 hindu_daily_motion: [126](#)
 hindu_date_occur: [126](#)
 hindu_day_count: [86](#)
 HINDU_EPOCH: [86](#), [126](#)
 hindu_equation_of_time: [126](#)
 hindu_fullmoon_from_fixed: [126](#)
 HINDU_LOCATION: [126](#)
 hindu_lunar_date: [126](#), [175](#)
 hindu_lunar_day: [126](#)
 hindu_lunar_day_at_or_after: [126](#)
 hindu_lunar_day_from_moment: [126](#)
 HINDU_LUNAR_ERA: [126](#)
 hindu_lunar_event: [126](#)

hindu_lunar_from_fixed: [126](#), [181](#)
 hindu_lunar_holiday: [126](#)
 hindu_lunar_leap_day: [126](#)
 hindu_lunar_leap_month: [126](#)
 hindu_lunar_longitude: [126](#)
 hindu_lunar_month: [126](#)
 hindu_lunar_new_year: [126](#)
 hindu_lunar_phase: [126](#)
 hindu_lunar_station: [126](#)
 hindu_lunar_year: [126](#)
 hindu_mean_position: [126](#)
 hindu_new_moon_before: [126](#)
 hindu_rising_sign: [126](#)
 HINDU_SIDEREAL_MONTH: [126](#)
 HINDU_SIDEREAL_YEAR: [126](#)
 hindu_sine: [126](#)
 hindu_sine_table: [126](#)
 hindu_solar_date: [86](#), [87](#), [126](#), [175](#)
 HINDU_SOLAR_ERA: [126](#)
 hindu_solar_from_fixed: [126](#), [181](#)
 hindu_solar_longitude: [126](#)
 hindu_solar_longitude_at_or_after: [126](#)
 hindu_solar_sidereal_difference: [126](#)
 hindu_sundial_time: [126](#)
 hindu_sunrise: [126](#)
 hindu_sunset: [126](#)
 HINDU_SYNODIC_MONTH: [126](#)
 hindu_tithi_occur: [126](#)
 hindu_tropical_longitude: [126](#)
 hindu_true_position: [126](#)
 hindu_zodiac: [126](#)
 hour: [39](#), [40](#), [42](#), [43](#), [95](#), [96](#), [97](#)
 hr: [80](#), [86](#), [92](#), [95](#), [97](#), [112](#), [115](#), [118](#), [123](#), [126](#), [132](#)
 IDES: [61](#)
 ides_of_month: [61](#)
 ifloor: [14](#), [15](#), [40](#), [46](#), [48](#), [86](#), [97](#), [112](#), [115](#), [120](#), [123](#), [126](#), [129](#), [132](#)
 independence_day: [56](#)
 interval: [28](#), [47](#), [56](#), [89](#), [126](#)
 invert_angular: [28](#), [28](#), [30](#), [99](#), [109](#), [126](#)
 iround: [15](#), [96](#), [109](#), [112](#), [115](#), [120](#), [123](#), [126](#), [132](#)
 is_arithmetic_french_leap_year: [120](#)
 is_arithmetic_persian_leap_year: [112](#)
 is_chinese_no_major_solar_term: [123](#)
 is_chinese_prior_leap_month: [123](#)
 is_coptic_leap_year: [71](#)
 is_gregorian_leap_year: [54](#), [55](#), [56](#), [57](#), [115](#)
 is_hebrew_leap_year: [80](#)
 is_hebrew_sabbatical_year: [80](#)
 is_hindu_expunged: [126](#)
 is_hindu_lunar_on_or_before: [126](#)
 is_in_range: [47](#), [56](#), [126](#)
 is_islamic_leap_year: [77](#)
 is_iso_long_year: [68](#)
 is_julian_leap_year: [59](#), [60](#), [61](#)
 IslamicAppendixCTestCase: [158](#)
 islamic_date: [77](#), [78](#), [132](#), [155](#)

ISLAMIC_EPOCH: [77](#), [132](#)
 islamic_from_fixed: [77](#), [78](#), [159](#)
 islamic_in_gregorian: [77](#)
 ISLAMIC_LOCATION: [132](#)
 IslamicSmokeTestCase: [78](#)
 is_long_marheshvan: [80](#)
 IsoAppendixCTestCase: [146](#)
 iso_date: [68](#), [69](#), [139](#)
 iso_day: [68](#)
 iso_from_fixed: [68](#), [69](#), [147](#)
 is_old_hindu_lunar_leap_year: [86](#)
 ISOSmokeTestCase: [69](#)
 iso_week: [68](#)
 iso_year: [68](#)
 is_short_kislev: [80](#)
 is_tibetan_leap_month: [129](#)
 IYYAR: [80](#)
 J2000: [92](#), [97](#), [99](#), [109](#)
 JAFFA: [94](#), [132](#)
 JANUARY: [53](#), [56](#), [57](#), [59](#), [92](#), [97](#), [123](#)
 japanese_location: [123](#)
 JD_EPOCH: [48](#)
 jd_from_fixed: [48](#), [143](#)
 jd_from_moment: [48](#)
 JERUSALEM: [92](#), [132](#), [186](#)
 jewish_dusk: [95](#)
 jewish_morning_end: [95](#)
 jewish_sabbath_ends: [95](#)
 jovian_year: [86](#), [87](#)
 JulianAppendixCTestCase: [149](#)
 julian_centuries: [92](#), [97](#), [99](#), [109](#), [110](#)
 julian_date: [59](#), [60](#), [61](#), [63](#), [71](#), [74](#), [77](#), [80](#), [83](#), [86](#), [112](#), [139](#), [164](#)
 JulianDayAppendixCTestCase: [142](#)
 JULIAN_EPOCH: [59](#)
 julian_from_fixed: [59](#), [60](#), [61](#), [63](#), [148](#)
 julian_in_gregorian: [63](#)
 JulianSmokeTestCase: [60](#)
 julian_year_from_auc_year: [61](#)
 JULY: [53](#), [56](#), [61](#), [77](#), [97](#), [123](#)
 JUNE: [53](#)
 kajeng_keliwon: [89](#)
 KALENDS: [60](#), [61](#), [62](#)
 karana: [126](#)
 kday_after: [56](#), [74](#), [132](#)
 kday_before: [56](#), [80](#)
 kday_nearest: [56](#)
 kday_on_or_after: [56](#), [126](#)
 kday_on_or_before: [56](#)
 KISLEV: [80](#), [81](#)
 korean_location: [123](#)
 korean_year: [123](#)
 labor_day: [56](#)
 last_day_of_hebrew_month: [80](#)
 last_kday: [56](#)
 last_month_of_hebrew_year: [80](#)
 LAST_QUARTER: [109](#)

latitude: [92](#), [95](#), [107](#), [109](#), [110](#), [126](#)
 list_range: [47](#), [63](#), [71](#), [77](#), [80](#), [126](#), [129](#)
 local_from_apparent: [92](#)
 local_from_standard: [92](#)
 local_from_universal: [92](#), [132](#)
 location: [92](#), [95](#), [110](#), [112](#), [115](#), [118](#), [123](#), [126](#), [132](#)
 longitude: [92](#), [97](#), [99](#), [109](#), [110](#), [123](#), [126](#), [184](#)
 losar: [129](#)
 lunar_altitude: [94](#), [110](#), [132](#)
 lunar_anomaly: [105](#), [106](#), [109](#), [110](#), [129](#)
 lunar_diameter: [110](#)
 lunar_distance: [110](#)
 lunar_elongation: [101](#), [102](#), [109](#), [110](#)
 lunar_latitude: [94](#), [109](#), [110](#), [132](#)
 lunar_longitude: [94](#), [109](#), [110](#), [186](#)
 lunar_node: [109](#)
 lunar_parallax: [110](#)
 lunar_perigee: [109](#)
 lunar_phase: [92](#), [109](#), [126](#), [132](#)
 lunar_phase_at_or_after: [109](#), [132](#)
 lunar_phase_at_or_before: [109](#)
 lunar_position: [110](#)
 lunar_true_node: [109](#)
 major_solar_term_on_or_after: [123](#), [177](#)
 MARCH: [53](#), [56](#), [57](#), [59](#), [61](#), [112](#), [115](#), [123](#)
 MARHESHVAN: [80](#)
 mawlid_an_nabi: [77](#)
 MAY: [53](#), [56](#), [61](#)
 MayanAppendixCTestCase: [162](#)
 mayan_baktun: [83](#)
 mayan_calendar_round_on_or_before: [83](#)
 MAYAN_EPOCH: [83](#)
 mayan_haab_date: [83](#), [84](#), [155](#)
 mayan_haab_day: [83](#)
 MAYAN_HAAB_EPOCH: [83](#)
 mayan_haab_from_fixed: [83](#), [84](#), [163](#)
 mayan_haab_month: [83](#)
 mayan_haab_on_or_before: [83](#), [84](#)
 mayan_haab_ordinal: [83](#)
 mayan_katun: [83](#)
 mayan_kin: [83](#)
 mayan_long_count_date: [83](#), [84](#), [155](#)
 mayan_long_count_from_fixed: [83](#), [84](#), [163](#)
 MayanSmokeTestCase: [84](#)
 mayan_tun: [83](#)
 mayan_tzolkin_date: [83](#), [84](#), [155](#)
 MAYAN_TZOLKIN_EPOCH: [83](#)
 mayan_tzolkin_from_fixed: [83](#), [84](#), [163](#)
 mayan_tzolkin_name: [83](#)
 mayan_tzolkin_number: [83](#)
 mayan_tzolkin_on_or_before: [83](#), [84](#)
 mayan_tzolkin_ordinal: [83](#)
 mayan_uinal: [83](#)
 mayan_year_bearer_from_fixed: [83](#)
 mean_lunar_longitude: [99](#), [100](#), [109](#)
 MEAN_SIDEREAL_YEAR: [97](#), [126](#)

MEAN_SYNODIC_MONTH: [97](#), [109](#), [123](#), [132](#)
 MEAN_TROPICAL_YEAR: [97](#), [99](#), [112](#), [115](#), [120](#), [123](#)
 MECCA: [92](#)
 memorial_day: [56](#)
 mesha_samkranti: [126](#)
 midday: [92](#), [95](#), [112](#)
 midday_in_tehran: [112](#)
 midnight: [92](#), [118](#), [123](#), [126](#)
 midnight_in_china: [123](#)
 midnight_in_paris: [118](#), [119](#), [120](#)
 minor_solar_term_on_or_after: [123](#)
 minute: [39](#), [40](#), [42](#), [43](#), [96](#), [126](#)
 MJD_EPOCH: [48](#)
 mjd_from_fixed: [48](#), [143](#)
 mod: [15](#), [28](#), [38](#), [40](#), [46](#), [54](#), [55](#), [59](#), [65](#), [71](#), [74](#), [77](#), [80](#), [83](#), [86](#), [89](#), [92](#), [95](#), [97](#), [99](#), [109](#),
[110](#), [112](#), [115](#), [120](#), [123](#), [126](#), [129](#), [132](#), [186](#), [242](#)
 ModernHinduAppendixCTestCase: [180](#)
 molad: [80](#)
 moment_from_jd: [48](#)
 moment_of_depression: [92](#)
 MONDAY: [38](#), [56](#), [80](#), [81](#), [139](#)
 moon_node: [107](#), [108](#), [109](#), [110](#)
 moonrise: [92](#)
 MORNING: [92](#)
 mt: [92](#), [110](#), [112](#), [115](#), [118](#), [123](#), [126](#), [132](#), [155](#), [163](#)
 naw_ruz: [112](#)
 NEW: [109](#), [132](#)
 new_moon_at_or_after: [109](#), [123](#), [126](#), [186](#)
 new_moon_before: [109](#), [123](#), [126](#)
 next: [16](#), [18](#), [80](#), [109](#), [112](#), [115](#), [120](#), [123](#), [126](#), [132](#), [184](#), [193](#), [200](#)
 nighttime_temporal_hour: [95](#)
 NISAN: [80](#), [132](#)
 NONES: [61](#)
 nones_of_month: [61](#)
 NOVEMBER: [53](#), [56](#), [57](#), [60](#), [62](#)
 nth_kday: [56](#), [68](#)
 nth_new_moon: [109](#)
 nutation: [97](#), [98](#), [109](#)
 obliquity: [92](#), [97](#)
 observational_hebrew_from_fixed: [132](#), [166](#), [243](#)
 observational_hebrew_new_year: [132](#)
 observational_islamic_from_fixed: [132](#), [159](#)
 OCTOBER: [53](#), [60](#), [61](#), [80](#), [119](#)
 odd: [89](#), [123](#)
 OldHinduAppendixCTestCase: [178](#)
 old_hindu_lunar_date: [86](#), [87](#), [175](#)
 old_hindu_lunar_day: [86](#)
 old_hindu_lunar_from_fixed: [86](#), [87](#), [179](#)
 old_hindu_lunar_leap: [86](#)
 old_hindu_lunar_month: [86](#)
 old_hindu_lunar_year: [86](#)
 OldHinduSmokeTestCase: [87](#)
 old_hindu_solar_from_fixed: [86](#), [87](#), [179](#)
 omer: [80](#)
 orthodox_easter: [74](#), [168](#)
 PARIS: [118](#), [186](#)

passover: [80](#)
 pentecost: [74](#)
 PersianAppendixCTestCase: [171](#)
 persian_date: [112](#)
 PERSIAN_EPOCH: [112](#)
 persian_from_fixed: [112](#), [172](#)
 persian_new_year_on_or_before: [112](#)
 phasis_on_or_after: [132](#)
 phasis_on_or_before: [132](#)
 poly: [34](#), [36](#), [92](#), [97](#), [99](#), [101](#), [103](#), [105](#), [107](#), [109](#), [110](#)
 positions_in_range: [89](#)
 possible_hebrew_days: [80](#), [81](#)
 precession: [99](#), [109](#), [126](#)
 purim: [80](#)
 qing_ming: [123](#)
 quotient: [14](#), [14](#), [55](#), [56](#), [59](#), [65](#), [68](#), [71](#), [74](#), [77](#), [80](#), [83](#), [86](#), [89](#), [112](#), [115](#), [120](#), [123](#), [126](#),
[132](#)
 radians_from_degrees: [92](#), [92](#)
 rama: [126](#)
 rd: [37](#), [38](#), [48](#), [52](#), [65](#), [68](#), [84](#), [94](#), [123](#), [139](#), [141](#), [143](#), [145](#), [147](#), [148](#), [150](#), [151](#), [154](#), [155](#),
[157](#), [159](#), [161](#), [163](#), [164](#), [166](#), [168](#), [170](#), [172](#), [174](#), [175](#), [177](#), [179](#), [181](#), [183](#), [184](#), [186](#), [243](#)
 refraction: [92](#), [110](#)
 right_ascension: [92](#), [94](#), [110](#)
 roman_count: [61](#)
 roman_date: [60](#), [61](#), [62](#), [139](#)
 roman_event: [61](#)
 roman_from_fixed: [60](#), [61](#), [62](#), [148](#)
 roman_leap: [61](#)
 roman_month: [61](#)
 RomanSmokeTestCase: [62](#)
 roman_year: [61](#)
 sacred_wednesdays: [126](#)
 sacred_wednesdays_in_range: [126](#)
 SATURDAY: [38](#), [80](#), [81](#), [139](#)
 sec: [92](#)
 seconds: [39](#), [42](#), [43](#), [46](#), [92](#), [96](#)
 secs: [92](#), [99](#)
 SEPTEMBER: [53](#), [56](#), [118](#)
 sh_ela: [80](#)
 SHEVAT: [80](#), [81](#)
 shift_days: [80](#)
 shiva: [126](#)
 sidereal_from_moment: [97](#), [110](#)
 sidereal_lunar_longitude: [109](#)
 sidereal_solar_longitude: [99](#), [126](#)
 SIDEREAL_START: [99](#), [109](#), [126](#)
 sidereal_zodiac: [126](#)
 sigma: [31](#), [33](#), [97](#), [109](#), [110](#)
 signum: [92](#), [97](#), [126](#)
 sin_degrees: [92](#), [97](#), [99](#), [109](#), [110](#), [126](#)
 sine_offset: [92](#)
 SIVAN: [80](#)
 solar_anomaly: [103](#), [104](#), [109](#), [110](#), [129](#)
 solar_longitude: [92](#), [95](#), [97](#), [99](#), [109](#), [112](#), [115](#), [120](#), [123](#), [126](#), [186](#)
 solar_longitude_after: [92](#), [99](#), [123](#), [132](#), [186](#)
 SPRING: [99](#), [112](#), [115](#), [132](#), [186](#)

standard_day: [39](#), [55](#), [56](#), [59](#), [61](#), [65](#), [71](#), [77](#), [80](#), [86](#), [112](#), [120](#), [126](#), [132](#)
 standard_from_local: [92](#)
 standard_from_sundial: [95](#), [126](#)
 standard_from_universal: [92](#), [123](#)
 standard_month: [39](#), [55](#), [56](#), [59](#), [61](#), [65](#), [71](#), [77](#), [80](#), [86](#), [112](#), [120](#), [126](#), [132](#)
 standard_year: [39](#), [55](#), [56](#), [59](#), [61](#), [63](#), [65](#), [71](#), [77](#), [80](#), [86](#), [112](#), [120](#), [126](#), [132](#), [145](#)
 start: [47](#), [56](#), [80](#), [89](#), [95](#), [115](#), [118](#), [123](#), [126](#), [132](#), [186](#)
 summa: [22](#), [24](#), [80](#)
 SUMMER: [99](#), [186](#)
 SUNDAY: [38](#), [56](#), [68](#), [74](#), [80](#), [132](#), [139](#)
 sunrise: [92](#), [95](#), [126](#), [242](#)
 sunset: [92](#), [95](#), [115](#), [126](#), [132](#), [184](#), [186](#)
 sunset_in_haifa: [115](#)
 ta_anit_esther: [80](#)
 TAMMUZ: [80](#)
 tangent_degrees: [92](#), [95](#), [97](#)
 TEHRAN: [112](#)
 testAltGregorian: [145](#)
 testAltSumma: [24](#)
 testArmenian: [151](#)
 testAztec: [163](#)
 testBahai: [161](#)
 testBalinese: [170](#)
 testBinarySearch: [27](#)
 testBirkathHaHama: [81](#)
 testChinese: [177](#)
 testClockFromMoment: [42](#)
 testCoptic: [154](#)
 testDawnInParis: [186](#)
 testDynamicalFromUniversal: [96](#)
 testEaster: [168](#)
 testEgyptian: [150](#)
 testEthiopic: [157](#)
 testFinal: [21](#)
 testFrenchRevolutionary: [174](#)
 testGregorian: [145](#)
 testHebrew: [166](#)
 testHinduLunisolarAstronomicalFromFixed: [181](#)
 testHinduLunisolarAstronomicalToFixed: [181](#)
 testHinduLunisolarModernFromFixed: [181](#)
 testHinduLunisolarModernToFixed: [181](#)
 testHinduSolarAstronomicalFromFixed: [181](#)
 testHinduSolarAstronomicalToFixed: [181](#)
 testHinduSolarModernFromFixed: [181](#)
 testHinduSolarModernToFixed: [181](#)
 testInvertAngular: [30](#)
 testIslamic: [159](#)
 testIso: [147](#)
 testJulian: [148](#)
 testJulianDay: [143](#)
 testLunarAnomaly: [106](#)
 testLunarElongation: [102](#)
 testLunarLongitude: [186](#)
 testMayan: [163](#)
 testMeanLunarLongitude: [100](#)
 testMoonNode: [108](#)

testNextNewMoon: [186](#)
 testNextSolsticeEquinox: [186](#)
 testNutation: [98](#)
 testOldHindu: [179](#)
 testPersian: [172](#)
 testPoly: [36](#)
 testPossibleHebrewDays: [81](#)
 testSigma: [33](#)
 testSolarAnomaly: [104](#)
 testSolarLongitude: [186](#)
 testSumma: [24](#)
 testSunsetInJerusalem: [186](#)
 testTibetan: [183](#)
 testTimeFromClock: [45](#)
 testTzomTevet: [81](#)
 testUniversalFromDynamical: [96](#)
 testUrbanaWinter: [93](#)
 testWeekdays: [141](#)
 TEVET: [80](#)
 THURSDAY: [38](#), [68](#), [80](#), [81](#), [139](#)
 TibetanAppendixCTestCase: [182](#)
 tibetan_date: [129](#), [175](#)
 tibetan_day: [129](#)
 tibetan_from_fixed: [129](#), [183](#)
 tibetan_leap_day: [129](#)
 tibetan_leap_month: [129](#)
 tibetan_month: [129](#)
 tibetan_moon_equation: [129](#)
 tibetan_new_year: [129](#)
 tibetan_sun_equation: [129](#)
 tibetan_year: [129](#)
 TimeAndAstronomySmokeTestCase: [94](#)
 time_from_clock: [43](#), [45](#), [96](#)
 time_from_moment: [40](#), [92](#)
 time_of_day: [39](#), [40](#)
 tishah_be_av: [80](#)
 TISHRI: [80](#), [132](#)
 topocentric_lunar_altitude: [92](#), [110](#)
 TUESDAY: [38](#), [56](#), [80](#), [81](#), [139](#)
 tumpek: [89](#)
 tzom_tevet: [80](#), [81](#)
 UJJAIN: [126](#)
 universal_from_dynamical: [95](#), [96](#), [98](#), [109](#)
 universal_from_local: [92](#), [126](#)
 universal_from_standard: [92](#), [95](#), [112](#), [115](#), [118](#), [123](#), [132](#)
 unlucky_fridays_in_range: [56](#)
 URBANA: [92](#)
 urbana_sunset: [92](#)
 urbana_winter: [92](#), [93](#)
 vietnamese_location: [123](#)
 visible_crescent: [132](#)
 WEDNESDAY: [38](#), [80](#), [81](#), [126](#), [139](#)
 WINTER: [92](#), [99](#), [123](#), [186](#)
 jahrzeit: [80](#)
 jahrzeit_in_gregorian: [80](#)
 YEAR_ROME_FOUNDED: [61](#)

yoga: 126
yom_ha_zikkaron: 80
yom_kippur: 80
zone: 92, 123
zone_from_longitude: 92