calendar — Work with Dates

Purpose: The calendar module implements classes for working with dates to manage year/month/week oriented values.

The `calendar` module defines the `Calendar` class, which encapsulates calculations for values such as the dates of the weeks in a given month or year. In addition, the `TextCalendar` and `HTMLCalendar` classes can produce pre-formatted output.

Formatting Examples

The `prmonth()` method is a simple function that produces the formatted text output for a month.

```
# calendar_textcalendar.py

import calendar

c = calendar.TextCalendar(calendar.SUNDAY)
c.prmonth(2017, 7)
```

The example configures `TextCalendar` to start weeks on Sunday, following the American convention. The default is to use the European convention of starting a week on Monday.

The output looks like:

```
$ python3 calendar_textcalendar.py

     July 2017
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

A similar HTML table can be produced with `HTMLCalendar` and `formatmonth()`. The rendered output looks roughly the same as the plain text version, but is wrapped with HTML tags. Each table cell has a class attribute corresponding to the day of the week, so the HTML can be styled through CSS.

To produce output in a format other than one of the available defaults, use `calendar` to calculate the dates and organize the values into week and month ranges, then iterate over the result. The `weekheader()`, `monthcalendar()`, and `yeardays2calendar()` methods of `Calendar` are especially useful for that.

Calling `yeardays2calendar()` produces a sequence of “month row” lists. Each list includes the months as another list of weeks. The weeks are lists of tuples made up of day number (1-31) and weekday number (0-6). Days that fall outside of the month have a day number of 0.

```
# calendar_yeardays2calendar.py

import calendar
import pprint

cal = calendar.Calendar(calendar.SUNDAY)

cal_data = cal.yeardays2calendar(2017, 3)
print('len(cal_data)      :', len(cal_data))

top_months = cal_data[0]
print('len(top_months)    :', len(top_months))

first_month = top_months[0]
print('len(first_month)   :', len(first_month))
```

```
print('first_month:')
pprint.pprint(first_month, width=65)
```

Calling `yeardays2calendar(2017, 3)` returns data for 2017, organized with three months per row.

```
$ python3 calendar_yeardays2calendar.py

len(cal_data)      : 4
len(top_months)    : 3
len(first_month)   : 5
first_month:
[[(1, 6), (2, 0), (3, 1), (4, 2), (5, 3), (6, 4), (7, 5)],
 [(8, 6), (9, 0), (10, 1), (11, 2), (12, 3), (13, 4), (14, 5)],
 [(15, 6), (16, 0), (17, 1), (18, 2), (19, 3), (20, 4), (21,
5)],
 [(22, 6), (23, 0), (24, 1), (25, 2), (26, 3), (27, 4), (28,
5)],
 [(29, 6), (30, 0), (31, 1), (0, 2), (0, 3), (0, 4), (0, 5)]]
```

This is equivalent to the data used by `formatyear()`.

```
# calendar_formatyear.py

import calendar

cal = calendar.TextCalendar(calendar.SUNDAY)
print(cal.formatyear(2017, 2, 1, 1, 3))
```

For the same arguments, `formatyear()` produces this output:

```
$ python3 calendar_formatyear.py

                                2017

      January               February               March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7            1  2  3  4            1  2  3  4
 8  9 10 11 12 13 14   5  6  7  8  9 10 11   5  6  7  8  9 10 11
15 16 17 18 19 20 21  12 13 14 15 16 17 18  12 13 14 15 16 17 18
22 23 24 25 26 27 28  19 20 21 22 23 24 25  19 20 21 22 23 24 25
29 30 31              26 27 28              26 27 28 29 30 31

       April                  May                   June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1      1  2  3  4  5  6                1  2  3
 2  3  4  5  6  7  8   7  8  9 10 11 12 13   4  5  6  7  8  9 10
 9 10 11 12 13 14 15  14 15 16 17 18 19 20  11 12 13 14 15 16 17
16 17 18 19 20 21 22  21 22 23 24 25 26 27  18 19 20 21 22 23 24
23 24 25 26 27 28 29  28 29 30 31           25 26 27 28 29 30
30

        July                  August              September
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
                   1         1  2  3  4  5                   1  2
 2  3  4  5  6  7  8   6  7  8  9 10 11 12   3  4  5  6  7  8  9
 9 10 11 12 13 14 15  13 14 15 16 17 18 19  10 11 12 13 14 15 16
16 17 18 19 20 21 22  20 21 22 23 24 25 26  17 18 19 20 21 22 23
23 24 25 26 27 28 29  27 28 29 30 31        24 25 26 27 28 29 30
30 31

      October               November              December
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
 1  2  3  4  5  6  7            1  2  3  4                   1  2
 8  9 10 11 12 13 14   5  6  7  8  9 10 11   3  4  5  6  7  8  9
15 16 17 18 19 20 21  12 13 14 15 16 17 18  10 11 12 13 14 15 16
22 23 24 25 26 27 28  19 20 21 22 23 24 25  17 18 19 20 21 22 23
29 30 31              26 27 28 29 30        24 25 26 27 28 29 30
                                            31
```

The `day_name`, `day_abbr`, `month_name`, and `month_abbr` module attributes useful for producing custom formatted output (i.e.

The day_name, day_abbr, month_name, and month_abbr module attributes useful for producing custom formatted output (i.e., to include links in the HTML output). They are automatically configured correctly for the current locale.

Locales

To produce a calendar formatted for a locale other than the current default, use `LocaleTextCalendar` or `LocaleHTMLCalendar`.

```
# calendar_locale.py

import calendar

c = calendar.LocaleTextCalendar(locale='en_US')
c.prmonth(2017, 7)

print()

c = calendar.LocaleTextCalendar(locale='fr_FR')
c.prmonth(2017, 7)
```

The first day of the week is not part of the locale settings, and the value is taken from the argument to the calendar class just as with the regular `TextCalendar`.

```
$ python3 calendar_locale.py

     July 2017
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31

    juillet 2017
Lu Ma Me Je Ve Sa Di
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

Calculating Dates

Although the calendar module focuses mostly on printing full calendars in various formats, it also provides functions useful for working with dates in other ways, such as calculating dates for a recurring event. For example, the Python Atlanta User's Group meets on the second Thursday of every month. To calculate the dates for the meetings for a year, use the return value of `monthcalendar()`.

```
# calendar_monthcalendar.py

import calendar
import pprint

pprint.pprint(calendar.monthcalendar(2017, 7))
```

Some days have a 0 value. Those are days of the week that overlap with the given month, but that are part of another month.

```
$ python3 calendar_monthcalendar.py

[[0, 0, 0, 0, 0, 1, 2],
 [3, 4, 5, 6, 7, 8, 9],
 [10, 11, 12, 13, 14, 15, 16],
 [17, 18, 19, 20, 21, 22, 23],
 [24, 25, 26, 27, 28, 29, 30],
 [31, 0, 0, 0, 0, 0, 0]]
```

The first day of the week defaults to Monday. It is possible to change that by calling `setfirstweekday()`, but since the calendar module includes constants for indexing into the date ranges returned by `monthcalendar()`, it is more convenient to skip that step in this case.

To calculate the group meeting dates for a year, assuming they are always on the second Thursday of every month, look at the output of `monthcalendar()` to find the dates on which Thursdays fall. The first and last week of the month are padded with 0 values as placeholders for the days falling in the preceding or subsequent month. For example, if a month starts on a Friday, the value in the first week in the Thursday position will be 0.

```
# calendar_secondthursday.py

import calendar
import sys

year = int(sys.argv[1])

# Show every month
for month in range(1, 13):

    # Compute the dates for each week that overlaps the month
    c = calendar.monthcalendar(year, month)
    first_week = c[0]
    second_week = c[1]
    third_week = c[2]

    # If there is a Thursday in the first week,
    # the second Thursday is # in the second week.
    # Otherwise, the second Thursday must be in
    # the third week.
    if first_week[calendar.THURSDAY]:
        meeting_date = second_week[calendar.THURSDAY]
    else:
        meeting_date = third_week[calendar.THURSDAY]

    print('{:>3}: {:>2}'.format(calendar.month_abbr[month],
                                meeting_date))
```

So the meeting schedule for the year is:

```
$ python3 calendar_secondthursday.py 2017

Jan: 12
Feb:  9
Mar:  9
Apr: 13
May: 11
Jun:  8
Jul: 13
Aug: 10
Sep: 14
Oct: 12
Nov:  9
Dec: 14
```

See also

- Standard library documentation for calendar
- `time` – Lower-level time functions.
- `datetime` – Manipulate date values, including timestamps and time zones.
- `locale` – Locale settings.

Quick Links
Formatting Examples
Locales
Calculating Dates

This page was last updated 2018-03-18.

Navigation
datetime — Date and Time Value Manipulation
Mathematics

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example