Python Essentials 2:
Module 4
Miscellaneous

In this module, you will learn about:

- Generators, iterators and closures;
- Working with file-system, directory tree and files;
- Selected Python Standard Library modules (os, datetime, time, and calendar.)

Introduction to the calendar module

In addition to the datetime and time modules, the Python standard library provides a module called calendar which, as the name suggests, offers calendar-related functions.

One of them is of course displaying the calendar. It's important that the days of the week are displayed from Monday to Sunday, and each day of the week has its representation in the form of an integer:

Day of the week	Integer value	Constant
Monday	0	calendar.MONDAY
Tuesday	1	calendar.TUESDAY
Wednesday	2	calendar.WEDNESDAY
Thursday	3	calendar.THURSDAY
Friday	4	calendar.FRIDAY
Saturday	5	calendar.SATURDAY
Sunday	6	calendar.SUNDAY

The table above shows the representation of the days of the week in the calendar module. The first day of the week (Monday) is represented by the value 0 and the calendar.MONDAY constant, while the last day of the week (Sunday) is represented by the value 6 and the calendar.SUNDAY constant.

For months, integer values are indexed from 1, i.e., January is represented by 1, and December by 12. Unfortunately, there aren't constants that express the months.

The above information will be useful to you when working with the calendar module in this part of the course, but first let's start with some simple calendar examples.

Your first calendar

You will start your adventure with the calendar module with a simple function called calendar, which allows you to display the calendar for the whole year. Let's look at how to use it to display the calendar for 2020. Run the code in the editor and see what happens.
The result displayed is similar to the result of the cal command available in Unix. If you want to change the default calendar formatting, you can use the following parameters:

- w – date column width (default 2)
- l – number of lines per week (default 1)
- c – number of spaces between month columns (default 6)
- m – number of columns (default 3)

The calendar function requires you to specify the year, while the other parameters responsible for formatting are optional. We encourage you to try these parameters yourself.
A good alternative to the above function is the function called prcal, which also takes the same parameters as the calendar function, but doesn't require the use of the print function to display the calendar. Its use looks like this:

```
import calendar
calendar.prcal(2020)
```

Calendar for a specific month

The calendar module has a function called month, which allows you to display a calendar for a specific month. Its use is really simple, you just need to specify the year and month - check out the code in the editor.

```
import calendar
print(calendar.month(2020, 11))
```

The example displays the calendar for November 2020. As in the calendar function, you can change the default formatting using the following parameters:

- w – date column width (default 2)
- l – number of lines per week (default 1)

Note: You can also use the prmonth function, which has the same parameters as the month function, but doesn't require you to use the print function to display the calendar.

The setfirstweekday() function

As you already know, by default in the calendar module, the first day of the week is Monday. However, you can change this behavior using a function called setfirstweekday.

Do you remember the table showing the days of the week and their representation in the form of integer values? It's time to use it, because the setfirstweekday method requires a parameter expressing the day of the week in the form of an integer value. Take a look at the example in the editor.

```
import calendar

calendar.setfirstweekday(calendar.SUNDAY)
calendar.prmonth(2020, 12)
```

The example uses the calendar.SUNDAY constant, which contains a value of 6. Of course, you could pass this value directly to the setfirstweekday function, but the version with a constant is more elegant.

As a result, we get a calendar showing the month of December 2020, in which the first day of all the weeks is Sunday.

The weekday() function

Another useful function provided by the calendar module is the function called weekday, which returns the day of the week as an integer value for the given year, month, and day. Let's see it in practice.

Run the code in the editor to check the day of the week that falls on December 24, 2020.

```
import calendar
print(calendar.weekday(2020, 12, 24))
```

Result:

```
3
output
```

The weekday function returns 3, which means that December 24, 2020 is a Thursday.

The weekheader() function

You've probably noticed that the calendar contains weekly headers in a shortened form. If needed, you can get short weekday names using the weekheader method.

The weekheader method requires you to specify the width in characters for one day of the week. If the width you provide is greater than 3, you'll still get the abbreviated weekday names consisting of three characters.

So let's look at how to get a smaller header. Run the code in the editor.

```
import calendar
print(calendar.weekheader(2))
```

Result:

```
Mo Tu We Th Fr Sa Su
output
```

Note: If you change the first day of the week, e.g., using the setfirstweekday function, it'll affect the result of the weekheader function.

How do we check if a year is a leap year?

The calendar module provides two useful functions to check whether years are leap years.

The first one, called isleap, returns True if the passed year is leap, or False otherwise. The second one, called leapdays, returns the number of leap years in the given range of years.

Run the code in the editor.

```
import calendar

print(calendar.isleap(2020))
print(calendar.leapdays(2010, 2021))  # Up to but not including 2021.
```

Result:

```
True
3
output
```

In the example, we obtain the result 3, because in the period from 2010 to 2020 there are only three leap years (note: 2021 is not included). They are the years 2012, 2016, and 2020.

Classes for creating calendars

The presented functions aren't everything the calendar module offers. In addition to them, we can use the following classes:

- calendar.Calendar – provides methods to prepare calendar data for formatting;
- calendar.TextCalendar – is used to create regular text calendars;
- calendar.HTMLCalendar – is used to create HTML calendars;
- calendar.LocalTextCalendar – is a subclass of the calendar.TextCalendar class. The constructor of this class takes the locale parameter, which is used to return the appropriate months and weekday names.
- calendar.LocalHTMLCalendar – is a subclass of the calendar.HTMLCalendar class. The constructor of this class takes the locale parameter, which is used to return the appropriate months and weekday names.

During this course, you've already had the opportunity to create text calendars when discussing the functions of the calendar module.

Time to try something new. Let's take a closer look at the methods of the calendar class.

Creating a Calendar object

The Calendar class constructor takes one optional parameter named firstweekday, by default equal to 0 (Monday).

The firstweekday parameter must be an integer between 0-6. For this purpose, we can use the already-known constants - look at the code in the editor.

```
import calendar

c = calendar.Calendar(calendar.SUNDAY)

for weekday in c.iterweekdays():
   print(weekday, end=" ")
```

The program will output the following result:

```
6 0 1 2 3 4 5
output
```

The code example uses the Calendar class method named iterweekdays, which returns an iterator for week day numbers.

The first value returned is always equal to the value of the firstweekday property. Because in our example the first value returned is 6, it means that the week starts on a Sunday.

The itermonthdates() method

The Calendar class has several methods that return an iterator. One of them is the itermonthdates method, which requires specifying the year and month.

As a result, all days in the specified month and year are returned, as well as all days before the beginning of the month or the end of the month that are necessary to get a complete week.

Each day is represented by a datetime.date object. Take a look at the example in the editor.

```
import calendar

c = calendar.Calendar()

for date in c.itermonthdates(2019, 11):
   print(date, end=" ")
```

The code displays all days in November 2019. Because the first day of November 2019 was a Friday, the following days are also returned to get the complete week: 10/28/2019 (Monday) 10/29/2019 (Tuesday) 10/30/2019 (Wednesday) 10/31/2019 (Thursday).

The last day of November 2019 was a Saturday, so in order to keep the complete week, one more day is returned 12/01/2019 (Friday).

Other methods that return iterators

Another useful method in the Calendar class is the method called itermonthdates, which takes year and month as parameters, and then returns the iterator to the days of the week represented by numbers.

Take a look at the example in the editor.

```
import calendar

c = calendar.Calendar()

for iter in c.itermonthdays(2019, 11):
   print(iter, end=" ")
```

You'll have certainly noticed the large number of 0s returned as a result of the example code. These are days outside the specified month range that are added to keep the complete week.

The first four zeros represent 10/28/2019 (Monday) 10/29/2019 (Tuesday) 10/30/2019 (Wednesday) 10/31/2019 (Thursday). The remaining numbers are days in the month, except the last value of 0, which replaces the date 12/01/2019 (Sunday).

There are four other similar methods in the Calendar class that differ in data returned:

- itermonthdates2 – returns days in the form of tuples consisting of a day of the month number and a week day number;
- itermonthdates3 – returns days in the form of tuples consisting of a year, a month, and a day of the month numbers. This method has been available since version 3.7;
- itermonthdates4 – returns days in the form of tuples consisting of a year, a month, a day of the month, and a day of the week numbers. This method has been available since Python version 3.7.

For testing purposes, use the example above and see how the return values of the described methods look in practice.

The monthdays2calendar() method

The Calendar class has several other useful methods that you can learn more about in the documentation (https://docs.python.org/3/library/calendar.html).

One of them is the monthdays2calendar method, which takes the year and month, and then returns a list of weeks in a specific month. Each week is a tuple consisting of day numbers and weekday numbers. Look at the code in the editor.

```
import calendar

c = calendar.Calendar()

for data in c.monthdays2calendar(2020, 12):
    print(data)
```

Note that the days numbers outside the month are represented by 0, while the weekday numbers are a number from 0-6, where 0 is Monday and 6 is Sunday.

In a moment, this method may be useful for you to complete a laboratory task. Are you ready?

LAB

Estimated time
30-60 minutes

Level of difficulty
Easy

Objectives
Improving the student's skills in using the Calendar class.
Scenario
During this course, we looked at the Calendar class a bit. Your task is to extend its functionality with a new method called count_weekday_in_year, which takes a year and a weekday as parameters, and then returns the number of occurrences of a specific weekday in the year.

Use the following tips:

Create a class called MyCalendar that extends the Calendar class;
create the count_weekday_in_year method with the year and weekday parameters. The weekday parameter should be a value between 0-6, where 0 is Monday and 6 is Sunday. The method should return the number of days as an integer;
in your implementation, use the monthdays2calendar method of the Calendar class.
The following are the expected results:

Sample arguments

year=2019, weekday=0

Expected output

52

Sample arguments

year=2000, weekday=6

Expected output

53

Key takeaways

1. In the calendar module, the days of the week are displayed from Monday to Sunday. Each day of the week has its representation in the form of an integer, where the first day of the week (Monday) is represented by the value 0, while the last day of the week (Sunday) is represented by the value 6.

2. To display a calendar for any year, call the calendar function with the year passed as its argument, e.g.:

```
import calendar
print(calendar.calendar(2020))
```

Note: A good alternative to the above function is the function called prcal, which also takes the same parameters as the calendar function, but doesn't require the use of the print function to display the calendar.

3. To display a calendar for any month of the year, call the month function, passing year and month to it. For example:

```
import calendar
print(calendar.month(2020, 9))
```

Note: You can also use the prmonth function, which has the same parameters as the month function, but doesn't require the use of the print function to display the calendar.

4. The setfirstweekday function allows you to change the first day of the week. It takes a value from 0 to 6, where 0 is Sunday and 6 is Saturday.

5. The result of the weekday function is a day of the week as an integer value for a given year, month, and day:

```
import calendar
print(calendar.weekday(2020, 9, 29)) # This displays 1, which means Tuesday.
```

6. The weekheader function returns the weekday names in a shortened form. The weekheader method requires you to specify the width in characters for one day of the week. If the width you provide is greater than 3, you'll still get the abbreviated weekday names consisting of only three characters. For example:

```
import calendar
print(calendar.weekheader(2)) # This display: Mo Tu We Th Fr Sa Su
```

7. A very useful function available in the calendar module is the function called isleap, which, as the name suggests, allows you to check whether the year is a leap year or not:

```
import calendar
print(calendar.isleap(2020)) # This displays: True
```

8. You can create a calendar object yourself using the Calendar class, which, when creating its object, allows you to change the first day of the week with the optional firstweekday parameter, e.g.:

```
import calendar

c = calendar.Calendar(2)

for weekday in c.iterweekdays():
   print(weekday, end=" ")
# Result: 2 3 4 5 6 0 1
```

The iterweekdays returns an iterator for weekday numbers. The first value returned is always equal to the value of the firstweekday property.

Exercise 1

What is the output of the following snippet?

```
import calendar
print(calendar.weekheader(1))
```

Check
M T W T F S S

Exercise 2

What is the output of the following snippet?

```
import calendar

c = calendar.Calendar()

for weekday in c.iterweekdays():
   print(weekday, end=" ")
```

Check
0 1 2 3 4 5 6

Congratulations! You have completed PE2: Module 4.

Well done! You've reached the end of Module 4 and completed a major milestone in your Python programming education. Here's a short summary of the objectives you've covered and got familiar with in Module 4:

- generators and iterators;
- list comprehensions;
- the lambda, map, and filter functions;
- closures;
- working with files (file streams, file processing, diagnosing stream problems)
- processing text and binary files;
- selected Python STL modules: os, datetime, time, and calendar.

You are now ready to take the module quiz and attempt the final challenge: Module 4 Test, which will help you gauge what you've learned so far.