# CERTIFIED ASSOCIATE IN PYTHON PROGRAMMING

BY: IMRAN

Module 4: MISCELLANEOUS
Part 4: The datetime module

## Getting the current local date and creating date objects

- from datetime import date
- today = date.today()
- print("Today:", today)
- print("Year:", today.year)
- print("Month:", today.month)
- print("Day:", today.day)

PAKISTAN PSEB SOFTWARE EXPORT BOARD

ImranNust imran_muet muhammad-imran-b7865495

## Creating a date object from a timestamp

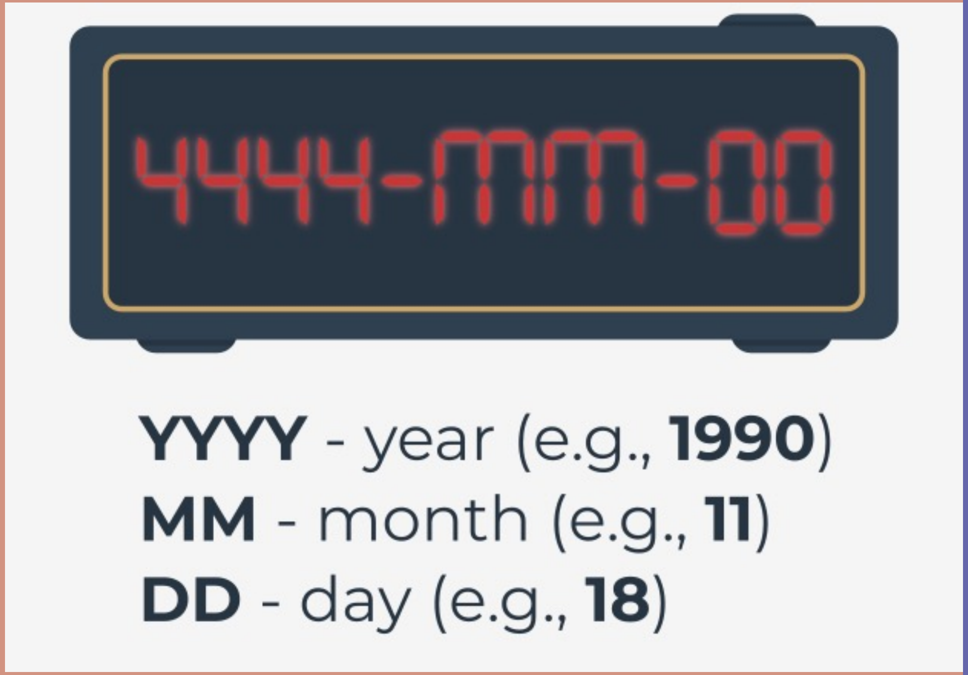
- The date class gives us the ability to create a *date* object from a *timestamp*.
- In Unix, the timestamp expresses the number of seconds since January 1, 1970, 00:00:00 (UTC). This date is called the **Unix epoch**, because this is when the counting of time began on Unix systems.
- The timestamp is actually the difference between a particular date (including time) and January 1, 1970, 00:00:00 (UTC), expressed in seconds.
- To create a date object from a timestamp, we must pass a Unix timestamp to the fromtimestamp method.
- For this purpose, we can use the time module, which provides time-related functions. One of them is a function called time() that returns the number of seconds from January 1, 1970 to the current moment in the form of a float number. Take a look at the example in the editor.

## Creating a date object from a timestamp

- **Run the code to see the output.**
- from datetime import date
- import time
- timestamp = time.time()
- print("Timestamp:", timestamp)
- d = date.fromtimestamp(timestamp)
- print("Date:", d)
- If you run the sample code several times, you'll be able to see how the timestamp increments itself. It's worth adding that the result of the time function depends on the platform, because **in Unix and Windows systems, leap seconds aren't counted**.
- **Note:** In this part of the course we'll also talk about the *time* module.

## Creating a date object using the ISO format

- The datetime module provides several methods to create a date object. One of them is the fromisoformat method, which takes a date in the **YYYY-MM-DD** format compliant with the ISO 8601 standard.
- The ISO 8601 standard defines how the date and time are represented. It's often used, so it's worth taking a moment to familiarize yourself with it. Take a look at the picture describing the values required by the format:



**YYYY** - year (e.g., **1990**)
**MM** - month (e.g., **11**)
**DD** - day (e.g., **18**)

Module 4: MISCELLANEOUS
Part 4: The datetime module

## Creating a date object using the ISO format

- Now look at the code:
- from datetime import date
- d = date.fromisoformat('2019-11-04')
- print(d)

- In our example, YYYY is 2019, MM is 11 (November), and DD is 04 (fourth day of November).
- When substituting the date, be sure to add 0 before a month or a day that is expressed by a number less than 10.
- **Note:** The fromisoformat method has been available in Python since version 3.7.

## The replace() method

- Sometimes you may need to replace the year, month, or day with a different value. You can't do this with the year, month, and day attributes because they're read-only. In this case, you can use the method named replace.
- Run the code.
- from datetime import date
- d = date(1991, 2, 5)
- print(d)
- d = d.replace(year=1992, month=1, day=16)
- print(d)
- Result: 1991-02-05 1992-01-16
- The *year*, *month*, and *day* parameters are optional. You can pass only one parameter to the replace method, e.g., *year*, or all three as in the example.
- The replace method returns a changed *date* object, so you must remember to assign it to some variable.

## What day of the week is it?

- One of the more helpful methods that makes working with dates easier is the method called weekday. It returns the day of the week as an integer, where 0 is Monday and 6 is Sunday. Run the
- from datetime import date
- d = date(2019, 11, 4)
- print(d.weekday())
- Result:
- 0

# What day of the week is it?

- The date class has a similar method called isoweekday, which also returns the day of the week as an integer, but 1 is Monday, and 7 is Sunday:

```
from datetime import date
d = date(2019, 11, 4)
print(d.isoweekday())
```

- Result:

```
1
```

- As you can see, for the same date we get a different integer, but expressing the same day of the week. The integer returned by the isodayweek method follows the ISO 85601 specification.

## Creating time objects

- You already know how to present a date using the date object. The datetime module also has a class that allows you to present time. Can you guess its name? Yes, it's called time:
- time(hour, minute, second, microsecond, *tzinfo, fold*)
- Run the following code:
- from datetime import time
- t = time(14, 53, 20, 1)
- print("Time:", t)
- print("Hour:", t.hour)
- print("Minute:", t.minute)
- print("Second:", t.second)
- print("Microsecond:", t.microsecond)

## The time module

- In addition to the time class, the Python standard library offers a module called time, which provides a time-related function. You already had the opportunity to learn the function called time when discussing the date class. Now we'll look at another useful function available in this module.
- You must spend many hours in front of a computer while doing this course. Sometimes you may feel the need to take a nap. Why not? Let's write a program that simulates a student's short nap.

# The time module

- Have a look at the code.

```
import time
class Student:
    def take_nap(self, seconds):
        print("I'm very tired. I have to take a nap. See you later.")
        time.sleep(seconds)
        print("I slept well! I feel great!")
student = Student()
student.take_nap(5)
```

- Result:
- I'm very tired. I have to take a nap. See you later. I slept well! I feel great!**output**

## The time module

- The most important part of the sample code is the use of the sleep function (yes, you may remember it from one of the previous labs earlier in the course), which suspends program execution for the given number of seconds.
- In our example it's 5 seconds. You're right, it's a very short nap.
- Extend the student's sleep by changing the number of seconds. Note that the sleep function accepts only an integer or a floating point number.

Module 4: MISCELLANEOUS
Part 4: The datetime module

## The ctime() function

- The time module provides a function called ctime, which **converts the time in seconds since January 1, 1970 (Unix epoch) to a string**.
- Do you remember the result of the time function? That's what you need to pass to ctime. Take a look at the example.
- import time
- timestamp = 1572879180
- print(time.ctime(timestamp))
- Result:
- Mon Nov 4 14:53:00 2019
- The ctime function returns a string for the passed timestamp. In our example, the timestamp expresses November 4, 2019 at 14:53:00.

## The ctime() function

- It's also possible to call the ctime function without specifying the time in seconds. In this case, the current time will be returned:
- `import time`
- `print(time.ctime())`

# The gmtime() and localtime() functions

- Some of the functions available in the time module require knowledge of the *struct_time* class, but before we get to know them, let's see what the class looks like:
  - `time.struct_time:`
  - `tm_year # specifies the year`
  - `tm_mon # specifies the month (value from 1 to 12)`
  - `tm_mday # specifies the day of the month (value from 1 to 31)`
  - `tm_hour # specifies the hour (value from 0 to 23)`
  - `tm_min # specifies the minute (value from 0 to 59)`
  - `tm_sec # specifies the second (value from 0 to 61 )`
  - `tm_wday # specifies the weekday (value from 0 to 6)`
  - `tm_yday # specifies the year day (value from 1 to 366)`

## The gmtime() and localtime() functions

- Some of the functions available in the time module require knowledge of the *struct_time* class, but before we get to know them, let's see what the class looks like:
  - `time.struct_time:`
  - `tm_isdst # specifies whether daylight saving time applies (1 – yes, 0 – no, -1 – it isn't known)`
  - `tm_zone # specifies the timezone name (value in an abbreviated form)`
  - `tm_gmtoff # specifies the offset east of UTC (value in seconds)`
- The *struct_time* class also allows access to values using indexes. Index 0 returns the value in *tm_year*, while 8 returns the value in *tm_isdst*.
- The exceptions are *tm_zone* and *tm_gmoff*, which cannot be accessed using indexes. Let's look at how to use the *struct_time* class in practice.

## The gmtime() and localtime() functions

- Run the code.
- import time
- timestamp = 1572879180
- print(time.gmtime(timestamp))
- print(time.localtime(timestamp))
- Result:
- time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53, tm_sec=0, tm_wday=0, tm_yday=308, tm_isdst=0)
  time.struct_time(tm_year=2019, tm_mon=11, tm_mday=4, tm_hour=14, tm_min=53, tm_sec=0, tm_wday=0, tm_yday=308, tm_isdst=0)
- The example shows two functions that convert the elapsed time from the Unix epoch to the *struct_time* object. The difference between them is that the gmtime function returns the *struct_time* object in UTC, while the localtime function returns local time. For the gmtime function, the *tm_isdst* attribute is always 0.

# The asctime() and mktime() functions

- The time module has functions that expect a *struct_time* object or a tuple that stores values according to the indexes presented when discussing the *struct_time* class. Run the example.
- import time
- timestamp = 1572879180
- st = time.gmtime(timestamp)
- print(time.asctime(st))
- print(time.mktime((2019, 11, 4, 14, 53, 0, 0, 308, 0)))
- Result:
- Mon Nov 4 14:53:00 2019 1572879180.0

## The asctime() and mktime() functions

- The first of the functions, called asctime, converts a *struct_time* object or a tuple to a string. Note that the familiar gmtime function is used to get the *struct_time* object. If you don't provide an argument to the asctime function, the time returned by the localtime function will be used.
- The second function called mktime converts a *struct_time* object or a tuple that expresses the local time to the number of seconds since the Unix epoch. In our example, we passed a tuple to it, which consists of the following values:

```
2019 => tm_year
11 => tm_mon
4 => tm_mday
14 => tm_hour
53 => tm_min
0 => tm_sec
0 => tm_wday
308 => tm_yday
0 => tm_isdst
```

## Creating datetime objects

- In the datetime module, date and time can be represented as separate objects or as one. The class that combines date and time is called datetime.
- datetime(year, month, day, hour, minute, second, microsecond, *tzinfo, fold)*
- Run the code:

```
from datetime import datetime
dt = datetime(2019, 11, 4, 14, 53)
print("Datetime:", dt)
print("Date:", dt.date())
print("Time:", dt.time())
```

- Result:

```
Datetime: 2019-11-04 14:53:00 Date: 2019-11-04 Time: 14:53:00
```

- The example creates a datetime object representing November 4, 2019 at 14:53:00. All parameters passed to the constructor go to read-only class attributes. They're *year*, *month*, *day*, *hour*, *minute*, *second*, *microsecond*, *tzinfo*, and *fold*.
- The example shows two methods that return two different objects. The method called date returns the *date* object with the given year, month, and day, while the method called time returns the *time* object with the given hour and minute.

## Methods that return the current date and time

- The datetime class has several methods that return the current date and time. These methods are:
  - today() — returns the current local date and time with the *tzinfo* attribute set to *None*;
  - now() — returns the current local date and time the same as the *today* method, unless we pass the optional argument *tz* to it. The argument of this method must be an object of the *tzinfo* subclass;
  - utcnow() — returns the current UTC date and time with the *tzinfo* attribute set to *None*.

```
from datetime import datetime
print("today:", datetime.today())
print("now:", datetime.now())
print("utcnow:", datetime.utcnow())
```

Module 4: MISCELLANEOUS
Part 4: The datetime module

## Getting a timesta mp

- There are many converters available on the Internet that can calculate a timestamp based on a given date and time, but how can we do it in the datetime module?
- This is possible thanks to the timestamp method provided by the datetime class. Look at the code in the editor.
- from datetime import datetime
- dt = datetime(2020, 10, 4, 14, 55)
- print("Timestamp:", dt.timestamp())
- Result:
- Timestamp: 1601823300.0
- The timestamp method returns a float value expressing the number of seconds elapsed between the date and time indicated by the *datetime* object and January 1, 1970, 00:00:00 (UTC).
-

## Date and time formatti ng

- All datetime module classes presented so far have a method called strftime. This is a very important method, because it allows us to return the date and time in the format we specify.
- The strftime method takes only one argument in the form of a string specifying the format that can consist of directives.
- A directive is a string consisting of the character % (percent) and a lowercase or uppercase letter, e.g., the directive %Y means the year with the century as a decimal number. Let's see it in an example. Run the code
- from datetime import date
- d = date(2020, 1, 4)
- print(d.strftime('%Y/%m/%d'))

## Date and time formatting

- Result:
- 2020/01/04
- In the example, we passed a format consisting of three directives separated by / (slash) to the strftime method. Of course, the separator character can be replaced by another character, or even by a string.
- You can put any characters in the format, but only recognizable directives will be replaced with the appropriate values. In our format we've used the following directives:
  - %Y – returns the year with the century as a decimal number. In our example, this is `2020`.
  - %m – returns the month as a zero-padded decimal number. In our example, it's `01`.
  - %d – returns the day as a zero-padded decimal number. In our example, it's `04`.

Module 4: MISCELLANEOUS
Part 4: The datetime module

## Date and time formatting

```
from datetime import time
from datetime import datetime
t = time(14, 53)
print(t.strftime("%H:%M:%S"))
dt = datetime(2020, 11, 4, 14, 53)
print(dt.strftime("%y/%B/%d %H:%M:%S"))
```

ImranNust imran_muet muhammad-imran-b7865495

## The strftime() function in the time module

- You probably won't be surprised to learn that the strftime function is available in the time module. It differs slightly from the strftime methods in the classes provided by the datetime module because, in addition to the format argument, it can also take (optionally) a tuple or struct_time object.
- If you don't pass a tuple or *struct_time* object, the formatting will be done using the current local time.

## The strftime() function in the time module

- Take a look at the example
- import time
- timestamp = 1572879180
- st = time.gmtime(timestamp)
- print(time.strftime("%Y/%m/%d %H:%M:%S", st))
- print(time.strftime("%Y/%m/%d %H:%M:%S"))
- Our result looks as follows:
- 2019/11/04 14:53:00 2020/10/12 12:19:40
- Creating a format looks the same as for the strftime methods in the datetime module. In our example, we use the %Y, %m, %d, %H, %M, and %S directives that you already know.

## Date and time operations

- Sooner or later you'll have to perform some calculations on the date and time. Fortunately, there's a class called timedelta in the datetime module that was created for just such a purpose.
- To create a timedelta object, just do subtraction on the date or datetime objects, just like we did in the example in the editor. Run it.
- from datetime import date
- from datetime import datetime
- d1 = date(2020, 11, 4)
- d2 = date(2019, 11, 4)
- print(d1 - d2)
- dt1 = datetime(2020, 11, 4, 0, 0, 0)
- dt2 = datetime(2019, 11, 4, 14, 53, 0)
- print(dt1 - dt2)
- Result:
- 366 days, 0:00:00 365 days, 9:07:00