**Example 1: Get Current Date and Time**

import datetime

datetime\_object = datetime.datetime.now()

print(datetime\_object)

When you run the program, the output will be something like:

2018-12-19 09:26:03.478039

Here, we have imported **datetime** module using import datetime statement.

One of the classes defined in the datetime module is datetime class. We then used now()method to create a datetime object containing the current local date and time.

**Example 2: Get Current Date**

import datetime

date\_object = datetime.date.today()

print(date\_object)

When you run the program, the output will be something like:

2018-12-19

In this program, we have used today() method defined in the date class to get a dateobject containing the current local date.

**What's inside datetime?**

We can use [dir()](https://www.programiz.com/python-programming/methods/built-in/dir) function to get a list containing all attributes of a module.

import datetime

print(dir(datetime))

When you run the program, the output will be:

['MAXYEAR', 'MINYEAR', '\_\_builtins\_\_', '\_\_cached\_\_', '\_\_doc\_\_', '\_\_file\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', '\_divide\_and\_round', 'date', 'datetime', 'datetime\_CAPI', 'time', 'timedelta', 'timezone', 'tzinfo']

Commonly used classes in the datetime module are:

* date Class
* time Class
* datetime Class
* timedelta Class

**datetime.date Class**

You can instantiate date objects from the date class. A date object represents a date (year, month and day).

**Example 3: Date object to represent a date**

import datetime

d = datetime.date(2019, 4, 13)

print(d)

When you run the program, the output will be:

2019-04-13

If you are wondering, date() in the above example is a constructor of the date class. The constructor takes three arguments: year, month and day.

The variable a is a date object.

We can only import date class from the datetime module. Here's how:

from datetime import date

a = date(2019, 4, 13)

print(a)

**Example 4: Get current date**

You can create a date object containing the current date by using a classmethod named today(). Here's how:

from datetime import date

today = date.today()

print("Current date =", today)

**Example 5: Get date from a timestamp**

We can also create date objects from a timestamp. A Unix timestamp is the number of seconds between a particular date and January 1, 1970 at UTC. You can convert a timestamp to date using fromtimestamp() method.

from datetime import date

import time

x = time.time()

print(x)#

timestamp = date.fromtimestamp(x)

print("Date =", timestamp)

When you run the program, the output will be:

Date = 2012-01-11

**Example 6: Print today's year, month and day**

We can get year, month, day, day of the week etc. from the date object easily. Here's how:

from datetime import date

# date object of today's date

today = date.today()

print("Current year:", today.year)

print("Current month:", today.month)

print("Current day:", today.day)

**datetime.time**

A time object instantiated from the time class represents the local time.

**Example 7: Time object to represent time**

from datetime import time

# time(hour = 0, minute = 0, second = 0)

a = time()

print("a =", a)

# time(hour, minute and second)

b = time(11, 34, 56)

print("b =", b)

# time(hour, minute and second)

c = time(hour = 11, minute = 34, second = 56)

print("c =", c)

# time(hour, minute, second, microsecond)

d = time(11, 34, 56, 234566)

print("d =", d)

When you run the program, the output will be:

a = 00:00:00

b = 11:34:56

c = 11:34:56

d = 11:34:56.234566

**Example 8: Print hour, minute, second and microsecond**

Once you create a time object, you can easily print its attributes such as hour, minute etc.

from datetime import time

a = time(11, 34, 56)

print("hour =", a.hour)

print("minute =", a.minute)

print("second =", a.second)

print("microsecond =", a.microsecond)

When you run the example, the output will be:

hour = 11

minute = 34

second = 56

microsecond = 0

Notice that we haven't passed microsecond argument. Hence, its default value 0 is printed.

**datetime.datetime**

The datetime module has a class named dateclass that can contain information from both **date** and **time** objects.

**Example 9: Python datetime object**

from datetime import datetime

#datetime(year, month, day)

a = datetime(2018, 11, 28)

print(a)

# datetime(year, month, day, hour, minute, second, microsecond)

b = datetime(2017, 11, 28, 23, 55, 59, 342380)

print(b)

When you run the program, the output will be:

2018-11-28 00:00:00

2017-11-28 23:55:59.342380

The first three arguments year, month and day in the datetime() constructor are mandatory.

**Example 10: Print year, month, hour, minute and timestamp**

from datetime import datetime

a = datetime(2017, 11, 28, 23, 55, 59, 342380)

print("year =", a.year)

print("month =", a.month)

print("hour =", a.hour)

print("minute =", a.minute)

print("timestamp =", a.timestamp())

When you run the program, the output will be:

year = 2017

month = 11

day = 28

hour = 23

minute = 55

timestamp = 1511913359.34238

**datetime.timedelta**

A timedelta object represents the difference between two dates or times.

**Example 11: Difference between two dates and times**

from datetime import datetime, date

t1 = date(year = 2018, month = 7, day = 12)

t2 = date(year = 2017, month = 12, day = 23)

t3 = t1 - t2

print("t3 =", t3)

t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)

t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)

t6 = t4 - t5

print("t6 =", t6)

print("type of t3 =", type(t3))

print("type of t6 =", type(t6))

When you run the program, the output will be:

t3 = 201 days, 0:00:00

t6 = -333 days, 1:14:20

type of t3 = <class 'datetime.timedelta'>

type of t6 = <class 'datetime.timedelta'>

Notice, both t3 and t6 are of <class 'datetime.timedelta'> type.

**Example 12: Difference between two timedelta objects**

from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)

t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)

t3 = t1 - t2

print("t3 =", t3)

When you run the program, the output will be:

t3 = 14 days, 13:55:39

Here, we have created two timedelta objects t1 and t2, and their difference is printed on the screen.

**Example 13: Printing negative timedelta object**

from datetime import timedelta

t1 = timedelta(seconds = 33)

t2 = timedelta(seconds = 54)

t3 = t1 - t2

print("t3 =", t3)

print("t3 =", abs(t3))

When you run the program, the output will be:

t3 = -1 day, 23:59:39

t3 = 0:00:21

**Example 14: Time duration in seconds**

You can get the total number of seconds in a timedelta object using total\_seconds() method.

from datetime import timedelta

t = timedelta(days = 5, hours = 1, seconds = 33, microseconds = 233423)

print("total seconds =", t.total\_seconds())

When you run the program, the output will be:

total seconds = 435633.233423

You can also find sum of two dates and times using + operator. Also, you can multiply and divide a timedelta object by integers and floats.

**Python format datetime**

The way date and time is represented may be different in different places, organizations etc. It's more common to use mm/dd/yyyy in the US, whereas dd/mm/yyyy is more common in the UK.

Python has strftime() and strptime() methods to handle this.

**Python strftime() - datetime object to string**

The strftime() method is defined under classes date, datetime and time. The method creates a formatted string from a given date, datetime or time object.

**Example 15: Format date using strftime()**

from datetime import datetime

# current date and time

now = datetime.now()

t = now.strftime("%H:%M:%S")

print("time:", t)

s1 = now.strftime("%m/%d/%Y, %H:%M:%S")

# mm/dd/YY H:M:S format

print("s1:", s1)

s2 = now.strftime("%d/%m/%Y, %H:%M:%S")

# dd/mm/YY H:M:S format

print("s2:", s2)

When you run the program, the output will be something like:

time: 04:34:52

s1: 12/26/2018, 04:34:52

s2: 26/12/2018, 04:34:52

Here, %Y, %m, %d, %H etc. are format codes. The strftime() method takes one or more format codes and returns a formatted string based on it.

In the above program, t, s1 and s2 are strings.

* %Y - year [0001,..., 2018, 2019,..., 9999]
* %m - month [01, 02, ..., 11, 12]
* %d - day [01, 02, ..., 30, 31]
* %H - hour [00, 01, ..., 22, 23
* %M - month [00, 01, ..., 58, 59]
* %S - second [00, 01, ..., 58, 59]

To learn more about strftime() and format codes, visit: [Python strftime()](https://www.programiz.com/python-programming/datetime/strftime).

**Python strptime() - string to datetime**

The strptime() method creates a datetime object from a given string (representing date and time).

**Example 16: strptime()**

from datetime import datetime

date\_string = "21 June, 2018"

print("date\_string =", date\_string)

date\_object = datetime.strptime(date\_string, "%d %B, %Y")

print("date\_object =", date\_object)

When you run the program, the output will be:

date\_string = 21 June, 2018

date\_object = 2018-06-21 00:00:00

The strptime() method takes two arguments:

1. a string representing date and time
2. format code equivalent to the first argument

The strftime() method returns a string representing date and time using [date](https://www.programiz.com/python-programming/datetime#date), [time](https://www.programiz.com/python-programming/datetime#time) or [datetime](https://www.programiz.com/python-programming/datetime" \l "datetime) object.

**Example 1: datetime to string using strftime()**

The program below converts a datetime object containing current date and time to different string formats.

from datetime import datetime

now = datetime.now() # current date and time

year = now.strftime("%Y")

print("year:", year)

month = now.strftime("%m")

print("month:", month)

day = now.strftime("%d")

print("day:", day)

time = now.strftime("%H:%M:%S")

print("time:", time)

date\_time = now.strftime("%m/%d/%Y, %H:%M:%S")

print("date and time:",date\_time)

When you run the program, the output will something like be:

year: 2018

month: 12

day: 24

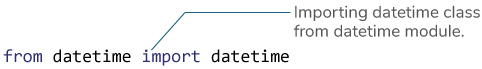
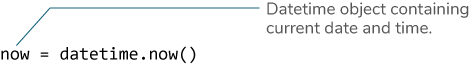
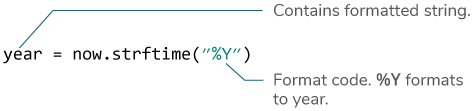
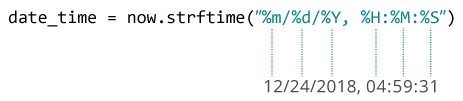
time: 04:59:31

date and time: 12/24/2018, 04:59:31

Here, year, day, time and date\_time are strings, whereas now is a datetime object.

**How strftime() works?**

In the above program, %Y, %m, %d etc. are format codes. The strftime() method takes one or more format codes as an argument and returns a formatted string based on it.

1. We imported datetime class from the datetime module. It's because the object of datetime class can access strftime() method.  
     
   
2. The datetime object containing current date and time is stored in now variable.  
     
   
3. The strftime() method can be used to create formatted strings.  
     
   
4. The string you pass to the strftime() method may contain more than one format codes.  
     
   

**Example 2: Creating string from a timestamp**

from datetime import datetime

timestamp = 1528797322

date\_time = datetime.fromtimestamp(timestamp)

print("Date time object:", date\_time)

d = date\_time.strftime("%m/%d/%Y, %H:%M:%S")

print("Output 2:", d)

d = date\_time.strftime("%d %b, %Y")

print("Output 3:", d)

d = date\_time.strftime("%d %B, %Y")

print("Output 4:", d)

d = date\_time.strftime("%I%p")

print("Output 5:", d)

When you run the program, the output will be:

Date time object: 2018-06-12 09:55:22

Output 2: 06/12/2018, 09:55:22

Output 3: 12 Jun, 2018

Output 4: 12 June, 2018

Output 5: 09AM

**Format Code List**

The table below shows all the codes that you can pass to the strftime() method.

| **Directive** | **Meaning** | **Example** |
| --- | --- | --- |
| %a | Abbreviated weekday name. | Sun, Mon, ... |
| %A | Full weekday name. | Sunday, Monday, ... |
| %w | Weekday as a decimal number. | 0, 1, ..., 6 |
| %d | Day of the month as a zero-padded decimal. | 01, 02, ..., 31 |
| %-d | Day of the month as a decimal number. | 1, 2, ..., 30 |
| %b | Abbreviated month name. | Jan, Feb, ..., Dec |
| %B | Full month name. | January, February, ... |
| %m | Month as a zero-padded decimal number. | 01, 02, ..., 12 |
| %-m | Month as a decimal number. | 1, 2, ..., 12 |
| %y | Year without century as a zero-padded decimal number. | 00, 01, ..., 99 |
| %-y | Year without century as a decimal number. | 0, 1, ..., 99 |
| %Y | Year with century as a decimal number. | 2013, 2019 etc. |
| %H | Hour (24-hour clock) as a zero-padded decimal number. | 00, 01, ..., 23 |
| %-H | Hour (24-hour clock) as a decimal number. | 0, 1, ..., 23 |
| %I | Hour (12-hour clock) as a zero-padded decimal number. | 01, 02, ..., 12 |
| %-I | Hour (12-hour clock) as a decimal number. | 1, 2, ... 12 |
| %p | Locale’s AM or PM. | AM, PM |
| %M | Minute as a zero-padded decimal number. | 00, 01, ..., 59 |
| %-M | Minute as a decimal number. | 0, 1, ..., 59 |
| %S | Second as a zero-padded decimal number. | 00, 01, ..., 59 |
| %-S | Second as a decimal number. | 0, 1, ..., 59 |
| %f | Microsecond as a decimal number, zero-padded on the left. | 000000 - 999999 |
| %z | UTC offset in the form +HHMM or -HHMM. |  |
| %Z | Time zone name. |  |
| %j | Day of the year as a zero-padded decimal number. | 001, 002, ..., 366 |
| %-j | Day of the year as a decimal number. | 1, 2, ..., 366 |
| %U | Week number of the year (Sunday as the first day of the week). All days in a new year preceding the first Sunday are considered to be in week 0. | 00, 01, ..., 53 |
| %W | Week number of the year (Monday as the first day of the week). All days in a new year preceding the first Monday are considered to be in week 0. | 00, 01, ..., 53 |
| %c | Locale’s appropriate date and time representation. | Mon Sep 30 07:06:05 2013 |
| %x | Locale’s appropriate date representation. | 09/30/13 |
| %X | Locale’s appropriate time representation. | 07:06:05 |
| %% | A literal '%' character. | % |

**Example 3: Locale's appropriate date and time**

from datetime import datetime

timestamp = 1528797322

date\_time = datetime.fromtimestamp(timestamp)

d = date\_time.strftime("%c")

print("Output 1:", d)

d = date\_time.strftime("%x")

print("Output 2:", d)

d = date\_time.strftime("%X")

print("Output 3:", d)

When you run the program, the output will be:

Output 1: Tue Jun 12 09:55:22 2018

Output 2: 06/12/18

Output 3: 09:55:22

Format codes %c, %x and %X are used for locale's appropriate date and time representation.

The strptime() method creates a [datetime](https://www.programiz.com/python-programming/datetime" \l "datetime) object from a given string.

**Note:** You cannot create datetime object from every string. The string needs to be in a certain format.

**Example 1: string to datetime object**

from datetime import datetime

date\_string = "21 June, 2018"

print("date\_string =", date\_string)

print("type of date\_string =", type(date\_string))

date\_object = datetime.strptime(date\_string, "%d %B, %Y")

print("date\_object =", date\_object)

print("type of date\_object =", type(date\_object))

When you run the program, the output will be:

date\_string = 21 June, 2018

type of date\_string = <class 'str'>

date\_object = 2018-06-21 00:00:00

type of date\_object = <class 'datetime.datetime'>

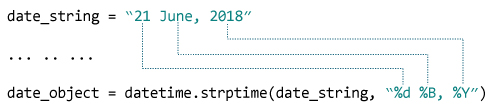
**How strptime() works?**

The strptime() class method takes two arguments:

* string (that be converted to datetime)
* format code

Based on the string and format code used, the method returns its equivalent datetimeobject.

In the above example:



Here,

* %d - Represents day of the month. **Example:** 01, 02, ..., 31
* %B - Month's name in full. **Example:** January, February etc.
* %Y - Year in four digits. **Example:** 2018, 2019 etc.

**Example 2: string to datetime object**

from datetime import datetime

dt\_string = "12/11/2018 09:15:32"

# Considering date is in dd/mm/yyyy format

dt\_object1 = datetime.strptime(dt\_string, "%d/%m/%Y %H:%M:%S")

print("dt\_object1 =", dt\_object1)

# Considering date is in mm/dd/yyyy format

dt\_object2 = datetime.strptime(dt\_string, "%m/%d/%Y %H:%M:%S")

print("dt\_object2 =", dt\_object2)

When you run the program, the output will be:

dt\_object1 = 2018-11-12 09:15:32

dt\_object2 = 2018-12-11 09:15:32

**Format Code List**

The table below shows all the format codes that you can use.

| **Directive** | **Meaning** | **Example** |
| --- | --- | --- |
| %a | Abbreviated weekday name. | Sun, Mon, ... |
| %A | Full weekday name. | Sunday, Monday, ... |
| %w | Weekday as a decimal number. | 0, 1, ..., 6 |
| %d | Day of the month as a zero-padded decimal. | 01, 02, ..., 31 |
| %-d | Day of the month as a decimal number. | 1, 2, ..., 30 |
| %b | Abbreviated month name. | Jan, Feb, ..., Dec |
| %B | Full month name. | January, February, ... |
| %m | Month as a zero-padded decimal number. | 01, 02, ..., 12 |
| %-m | Month as a decimal number. | 1, 2, ..., 12 |
| %y | Year without century as a zero-padded decimal number. | 00, 01, ..., 99 |
| %-y | Year without century as a decimal number. | 0, 1, ..., 99 |
| %Y | Year with century as a decimal number. | 2013, 2019 etc. |
| %H | Hour (24-hour clock) as a zero-padded decimal number. | 00, 01, ..., 23 |
| %-H | Hour (24-hour clock) as a decimal number. | 0, 1, ..., 23 |
| %I | Hour (12-hour clock) as a zero-padded decimal number. | 01, 02, ..., 12 |
| %-I | Hour (12-hour clock) as a decimal number. | 1, 2, ... 12 |
| %p | Locale’s AM or PM. | AM, PM |
| %M | Minute as a zero-padded decimal number. | 00, 01, ..., 59 |
| %-M | Minute as a decimal number. | 0, 1, ..., 59 |
| %S | Second as a zero-padded decimal number. | 00, 01, ..., 59 |
| %-S | Second as a decimal number. | 0, 1, ..., 59 |
| %f | Microsecond as a decimal number, zero-padded on the left. | 000000 - 999999 |
| %z | UTC offset in the form +HHMM or -HHMM. |  |
| %Z | Time zone name. |  |
| %j | Day of the year as a zero-padded decimal number. | 001, 002, ..., 366 |
| %-j | Day of the year as a decimal number. | 1, 2, ..., 366 |
| %U | Week number of the year (Sunday as the first day of the week). All days in a new year preceding the first Sunday are considered to be in week 0. | 00, 01, ..., 53 |
| %W | Week number of the year (Monday as the first day of the week). All days in a new year preceding the first Monday are considered to be in week 0. | 00, 01, ..., 53 |
| %c | Locale’s appropriate date and time representation. | Mon Sep 30 07:06:05 2013 |
| %x | Locale’s appropriate date representation. | 09/30/13 |
| %X | Locale’s appropriate time representation. | 07:06:05 |
| %% | A literal '%' character. | % |

**ValueError in strptime()**

If the string (first argument) and the format code (second argument) passed to the strptime() doesn't match, you will get ValueError. For example:

from datetime import datetime

date\_string = "12/11/2018"

date\_object = datetime.strptime(date\_string, "%d %m %Y")

print("date\_object =", date\_object)

If you run this program, you will get an error.

ValueError: time data '12/11/2018' does not match format '%d %m %Y'

here are a number of ways you can take to get current date. We will use date class of the [datetime](https://www.programiz.com/python-programming/datetime) module to accomplish this task.

## Example 1: Python get today's date

from datetime import date

today = date.today()

print("Today's date:", today)

Here, we imported date class from the datetime module. Then, we used date.today()method to get the current local date.

By the way, today variable will be a date object. You can use [strftime()](https://www.programiz.com/python-programming/datetime/strftime) method to create string representing date in different formats from this object.

## Example 2: Current date in different formats

from datetime import date

today = date.today()

# dd/mm/YY

d1 = today.strftime("%d/%m/%Y")

print("d1 =", d1)

# Textual month, day and year

d2 = today.strftime("%B %d, %Y")

print("d2 =", d2)

# mm/dd/y

d3 = today.strftime("%m/%d/%y")

print("d3 =", d3)

# Month abbreviation, day and year

d4 = today.strftime("%b-%d-%Y")

print("d4 =", d3)

When you run the program, the output will be something like:

d1 = 25/12/2018

d2 = December 25, 2018

d3 = 12/25/18

d4 = 12/25/18

If you need to get current date and time, you can use datetime class of the datetimemodule.

## Example 3: Get current date and time

from datetime import datetime

# datetime object containing current date and time

now = datetime.now()

print("now =", now)

# dd/mm/YY H:M:S

dt\_string = now.strftime("%d/%m/%Y %H:%M:%S")

print("date and time =", dt\_string)

Here, we have used datetime.now() to get current date and time. Then, we used strftime()method to create a string representing date and time in another format.

Python has a module named time to handle time-related tasks. To use functions defined in the module, we need to import the module first. Here's how:

import time

Here are commonly used time-related functions.

**Python time.time()**

The time() function returns the number of seconds passed since epoch.

For Unix system, January 1, 1970, 00:00:00 at **UTC** is epoch (the point where time begins).

import time

seconds = time.time()

print("Seconds since epoch =", seconds)

**Python time.ctime()**

The time.ctime() function takes seconds passed since epoch as an argument and returns a string representing local time.

import time

# seconds passed since epoch

seconds = 1545925769.9618232

local\_time = time.ctime(seconds)

print("Local time:", local\_time)

If you run the program, the output will be something like:

Local time: Thu Dec 27 15:49:29 2018

**Python time.sleep()**

The sleep() function suspends (delays) execution of the current thread for the given number of seconds.

import time

print("This is printed immediately.")

time.sleep(2.4)

print("This is printed after 2.4 seconds.")

To learn more, visit: [Python sleep()](https://www.programiz.com/python-programming/python-programming/date/sleep).

Before we talk about other time-related functions, let's explore time.struct\_time class in brief.

**time.struct\_time Class**

Several functions in the time module such as gmtime(), asctime() etc. either take time.struct\_time object as an argument or return it.

Here's an example of time.struct\_time object.

time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=27,

tm\_hour=6, tm\_min=35, tm\_sec=17,

tm\_wday=3, tm\_yday=361, tm\_isdst=0)

| Index | Attribute | Values |
| --- | --- | --- |
| 0 | tm\_year | 0000, ...., 2018, ..., 9999 |
| 1 | tm\_mon | 1, 2, ..., 12 |
| 2 | tm\_mday | 1, 2, ..., 31 |
| 3 | tm\_hour | 0, 1, ..., 23 |
| 4 | tm\_min | 0, 1, ..., 59 |
| 5 | tm\_sec | 0, 1, ..., 61 |
| 6 | tm\_wday | 0, 1, ..., 6; Monday is 0 |
| 7 | tm\_yday | 1, 2, ..., 366 |
| 8 | tm\_isdst | 0, 1 or -1 |

The values (elements) of the time.struct\_time object are accessible using both indices and attributes.

**Python time.localtime()**

The localtime() function takes the number of seconds passed since epoch as an argument and returns struct\_time in **local time**.

import time

result = time.localtime(1545925769)

print("result:", result)

print("\nyear:", result.tm\_year)

print("tm\_hour:", result.tm\_hour)

When you run the program, the output will be something like:

result: time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=27, tm\_hour=15, tm\_min=49, tm\_sec=29, tm\_wday=3, tm\_yday=361, tm\_isdst=0)

year: 2018

tm\_hour: 15

If no argument or None is passed to localtime(), the value returned by time() is used.

**Python time.gmtime()**

The gmtime() function takes the number of seconds passed since epoch as an argument and returns struct\_time in **UTC**.

import time

result = time.gmtime(1545925769)

print("result:", result)

print("\nyear:", result.tm\_year)

print("tm\_hour:", result.tm\_hour)

When you run the program, the output will be:

result = time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=28, tm\_hour=8, tm\_min=44, tm\_sec=4, tm\_wday=4, tm\_yday=362, tm\_isdst=0)

year = 2018

tm\_hour = 8

If no argument or None is passed to gmtime(), the value returned by time() is used.

**Python time.mktime()**

The mktime() function takes struct\_time (or a tuple containing 9 elements corresponding to struct\_time) as an argument and returns the seconds passed since epoch in local time. Basically, it's the inverse function of localtime().

import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

local\_time = time.mktime(t)

print("Local time:", local\_time)

The example below shows how mktime() and localtime() are related.

import time

seconds = 1545925769

# returns struct\_time

t = time.localtime(seconds)

print("t1: ", t)

# returns seconds from struct\_time

s = time.mktime(t)

print("\s:", seconds)

When you run the program, the output will be something like:

t1: time.struct\_time(tm\_year=2018, tm\_mon=12, tm\_mday=27, tm\_hour=15, tm\_min=49, tm\_sec=29, tm\_wday=3, tm\_yday=361, tm\_isdst=0)

s: 1545925769.0

**Python time.asctime()**

The asctime() function takes struct\_time (or a tuple containing 9 elements corresponding to struct\_time) as an argument and returns a string representing it. Here's an example:

import time

t = (2018, 12, 28, 8, 44, 4, 4, 362, 0)

result = time.asctime(t)

print("Result:", result)

When you run the program, the output will be:

Result: Fri Dec 28 08:44:04 2018

**Python time.strftime()**

The strftime() function takes struct\_time (or tuple corresponding to it) as an argument and returns a string representing it based on the format code used. For example,

import time

named\_tuple = time.localtime() # get struct\_time

time\_string = time.strftime("%m/%d/%Y, %H:%M:%S", named\_tuple)

print(time\_string)

When you run the program, the output will be something like:

12/28/2018, 09:47:41

Here, %Y, %m, %d, %H etc. are format codes.

* %Y - year [0001,..., 2018, 2019,..., 9999]
* %m - month [01, 02, ..., 11, 12]
* %d - day [01, 02, ..., 30, 31]
* %H - hour [00, 01, ..., 22, 23
* %M - month [00, 01, ..., 58, 59]
* %S - second [00, 01, ..., 58, 61]

To learn more, visit: [time.strftime()](https://docs.python.org/3/library/time.html" \l "time.strftime).

**Python time.strptime()**

The strptime() function parses a string representing time and returns struct\_time.

import time

time\_string = "21 June, 2018"

result = time.strptime(time\_string, "%d %B, %Y")

print(result)

When you run the program, the output will be:

time.struct\_time(tm\_year=2018, tm\_mon=6, tm\_mday=21, tm\_hour=0, tm\_min=0, tm\_sec=0, tm\_wday=3, tm\_yday=172, tm\_isdst=-1)

## What is Directory in Python?

If there are a large number of [files to handle](https://www.programiz.com/python-programming/file-operation) in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the os [module](https://www.programiz.com/python-programming/modules), which provides us with many useful methods to work with directories (and files as well).

## Get Current Directory

We can get the present working directory using the getcwd() method.

This method returns the current working directory in the form of a string. We can also use the getcwdb() method to get it as bytes object.

>>> import os

>>> os.getcwd()

'C:\\Program Files\\PyScripter'

>>> os.getcwdb()

b'C:\\Program Files\\PyScripter'

The extra backslash implies escape sequence. The print() function will render this properly.

>>> print(os.getcwd())

C:\Program Files\PyScripter

## Changing Directory

We can change the current working directory using the chdir() method.

The new path that we want to change to must be supplied as a string to this method. We can use both forward slash (/) or the backward slash (\) to separate path elements.

It is safer to use escape sequence when using the backward slash.

>>> os.chdir('C:\\Python33')

>>> print(os.getcwd())

C:\Python33

## List Directories and Files

All files and sub directories inside a directory can be known using the listdir() method.

This method takes in a path and returns a list of sub directories and files in that path. If no path is specified, it returns from the current working directory.

>>> print(os.getcwd())

C:\Python33

>>> os.listdir()

['DLLs',

'Doc',

'include',

'Lib',

'libs',

'LICENSE.txt',

'NEWS.txt',

'python.exe',

'pythonw.exe',

'README.txt',

'Scripts',

'tcl',

'Tools']

>>> os.listdir('G:\\')

['$RECYCLE.BIN',

'Movies',

'Music',

'Photos',

'Series',

'System Volume Information']

## Making a New Directory

We can make a new directory using the mkdir() method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

>>> os.mkdir('test')

>>> os.listdir()

['test']

## Renaming a Directory or a File

The rename() method can rename a directory or a file.

The first argument is the old name and the new name must be supplies as the second argument.

>>> os.listdir()

['test']

>>> os.rename('test','new\_one')

>>> os.listdir()

['new\_one']

## Removing Directory or File

A file can be removed (deleted) using the remove() method.

Similarly, the rmdir() method removes an empty directory.

>>> os.listdir()

['new\_one', 'old.txt']

>>> os.remove('old.txt')

>>> os.listdir()

['new\_one']

>>> os.rmdir('new\_one')

>>> os.listdir()

[]

However, note that rmdir() method can only remove empty directories.

In order to remove a non-empty directory we can use the rmtree() method inside the shutil module.

>>> os.listdir()

['test']

>>> os.rmdir('test')

Traceback (most recent call last):

...

OSError: [WinError 145] The directory is not empty: 'test'

>>> import shutil

>>> shutil.rmtree('test')

>>> os.listdir()

[]

# Default timezone is UTC and precisions is up to microseconds

#======================================

# Workflow

#======================================

# 1) Convert string to it's native datetime type.

# 2) Do all manipulation when the value is in datetime type

# 3) Convert it back to the type you need(if needed).

# This means that if you want to convert string "1970-01-01" to

# a 'datetime.datetime' type, you should first convert it to 'datetime.date' type:

# 1) Convert "1970-01-01" to 'datetime.date'

# 2) Convert from 'datetime.date' to 'datetime.datetime'

# This workflow clears up confusion as you won't be skipping around

# conversion steps, making it easy to create utility functions.

#======================================

# Explanation

#======================================

# To do any manipulation for date/datetime in string(like "1970-01-01"),

# you need to convert it to datetime type first

# What this means is that suppose you want to add 5 seconds to string "1970-01-01" and convert

# it to a datetime string like "1970-01-01 00:00:05.000000", you should use the following workflow:

# 1) Convert string "1970-01-01" to type 'datetime.date'

# 2) Convert it from 'datetime.date' to 'datetime.datetime'

# 3) Use timedelta to add 5 seconds

# 4) Use str(some\_datetime\_value) to convert it back to string.

# Using our utilities functions below, it would be something like this:

# date\_value = convert\_str\_to\_date("1970-01-01")

# datetime\_value = convert\_date\_to\_datetime\_start(date\_value)

# result = str(datetime\_addition(datetime\_value, 5, "seconds"))

# Suppose in above example, you want to add 2880 minutes(2 days equivalent)

# instead and get the result back in a date string like "1970-01-03",

# you should use the following workflow:

# 1) Convert string "1970-01-01" to type 'datetime.date'

# 2) Use timedelta to add 2880 minutes

# 4) Use str(some\_date\_value) to convert it back to string.

# Using our utilities functions below, it would be something like this:

# date\_value = convert\_str\_to\_date("1970-01-01")

# result = str(datetime\_addition(date\_value, 2880, "minutes"))

#======================================

# Date/Datetime Conversion

#======================================

"""

Default Timezone: UTC

Precision: Microseconds

"""

import pytz

from datetime import date, timedelta, datetime, time

def start\_time():

return time(0, 00, 00, 000000)

def end\_time():

return time(23, 59, 59, 999999)

def convert\_str\_to\_date(date\_str, format="%Y-%m-%d"):

"""

Given date in string and format, convert it to a 'datetime.date' type.

Return is datetime naive

"""

return datetime.strptime(date\_str, format)

def convert\_str\_to\_datetime(date\_str, format="%Y-%m-%d %H:%M:%S.%f"):

"""

Note: %z directive is only supported from Python 3.2 onwards.

Given datetime in string and format, convert it to a 'datetime.datetime' type.

Return is datetime naive

"""

return datetime.strptime(date\_str, format)

def convert\_date\_to\_datetime\_start(date\_value):

"""

Description

----------

Given date , convert it to datetime at end of day. Precision is up to seconds.

Return is datetime naive

Example

----------

From "1970-01-01" to "1970-01-01 00:00:00.000000"

Parameters

----------

date\_value: <type 'datetime.date'>

Returns

---------

<type 'datetime.datetime'>

Note

----------

Provided date\_value must be a 'datetime.date' type, if you have date in string,

use function:convert\_str\_to\_date to convert it 'datetime.date' first.

To avoid confusion , it is best to avoid skipping conversion steps.

"""

return datetime.combine(date\_value, start\_time())

def convert\_date\_to\_datetime\_end(date\_value):

"""

Description

----------

Given date , convert it to datetime at end of day. Precision is up to seconds.

Return is datetime naive

Example

----------

From "1970-01-01" to "1970-01-01 23:59:59:999999"

Parameters

----------

date\_value: <type 'datetime.date'>

Returns

---------

<type 'datetime.datetime'>

Note

----------

Provided date\_value must be a 'datetime.date' type, if you have date in string,

use function:convert\_str\_to\_date to convert it 'datetime.date' first.

To avoid confusion , it is best to avoid skipping conversion steps.

"""

return datetime.combine(date\_value, end\_time())

def convert\_datetime\_to\_date(datetime\_value):

"""

Convert datetime to date

From <type 'datetime.datetime'> to <type 'datetime.date'>

"""

return datetime\_value.date()

#======================================

# Timezone Functions

#======================================

def add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone='UTC'):

return pytz.timezone(timezone).localize(datetime\_naive)

def convert\_datetime\_aware\_to\_datetime\_naive(datetime\_aware):

return datetime\_aware.replace(tzinfo=None)

def convert\_datetime\_to\_another\_timezone(datetime\_value, timezone):

"""

Description

----------

Given timezone aware datetime object, convert it to another timezone.

Example

----------

From "1970-01-01 00:00:00.000000+0000" to "1970-01-01 08:00:00.000000+0800"

Parameters

----------

datetime\_value: <type 'datetime.datetime'>

timezone: string (but must be valid timezone for pytz)

Returns

---------

<type 'datetime.datetime'>

"""

return datetime\_value.astimezone(pytz.timezone(timezone))

def utc\_now\_in\_date():

return datetime.now(pytz.utc).date()

def utc\_now\_in\_datetime():

return datetime.now(pytz.utc)

#======================================

# Addition/Substraction Date/Datetime

#======================================

# Use timedelta to perform addition/subtraction in date/datetime types

def datetime\_addition(datetime\_to\_add, interval\_num, interval\_type):

"""

Description

----------

Generic function for datetime addition.

Example

----------

Given "1970-01-01 00:00:00:000000+0000" as datetime\_to\_add, 5 as interval\_num,

"seconds" as interval\_type, return "1970-01-01 00:00:05:000000+0000"

Parameters

----------

datetime\_to\_add: Any type in <type 'datetime.\*'> class.

interval\_num: integer

interval\_type: string

Returns

---------

Depending on datetime\_to\_add you pass in. Will return specific type from <type 'datetime.\*'> class

Note

--------

This function accepts any type in datetime class.

When you pass in <type 'datetime.date'> type like "1970-01-01"

and try something like adding 2880 minutes(2 days equivalent), you will get "1970-01-03"

"""

params = {interval\_type: interval\_num}

return datetime\_to\_add + timedelta(\*\*params)

def datetime\_subtraction(datetime\_to\_subtract, interval\_num, interval\_type):

"""

Description

----------

Generic function for datetime sustraction.

Example

----------

Given "1970-01-01" as datetime\_to\_subtract, 5 as interval\_num, "days" as interval\_type,

return "1969-12-27"

Parameters

----------

datetime\_to\_subtract: Any type in <type 'datetime.\*'> class.

interval\_num: integer

interval\_type: string

Returns

---------

Depending on datetime\_to\_subtract you pass in.

Will return specific type from <type 'datetime.\*'> class

Note

--------

This function accepts any type in datetime class.

When you pass in <type 'datetime.date'> type like "1970-01-03"

and try something like subtracting 2880 minutes(2 days equivalent), you will get "1970-01-01"

"""

params = {interval\_type: interval\_num}

return datetime\_to\_subtract - timedelta(\*\*params)

def add\_days\_to\_date(date\_to\_add, days\_in\_number):

return datetime\_addition(date\_to\_add, days\_in\_number, "days")

def add\_days\_to\_datetime(datetime\_to\_add, days\_in\_number):

return datetime\_addition(datetime\_to\_add, days\_in\_number, "days")

def subtract\_days\_from\_date(date\_to\_subtract, days\_in\_number):

return datetime\_subtraction(date\_to\_subtract, days\_in\_number, "days")

def subtract\_days\_from\_datetime(datetime\_to\_subtract, days\_in\_number):

return datetime\_subtraction(datetime\_to\_subtract, days\_in\_number, "days")

#======================================

# Utilities

#======================================

def last\_day\_of\_month(any\_day):

"""

Taken from http://stackoverflow.com/a/13565185/1446284

"""

next\_month = any\_day.replace(day=28) + timedelta(days=4) # this will never fail

return next\_month - timedelta(days=next\_month.day)

#======================================

# Ready-Made Functions

#======================================

def get\_today(timezone='UTC'):

return datetime.now(pytz.timezone(timezone)).date()

def get\_today\_start\_in\_datetime(timezone='UTC'):

"""

Return start of today in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

today = get\_today(timezone)

datetime\_naive = datetime.combine(today, start\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def get\_today\_end\_in\_datetime(timezone='UTC'):

"""

Return end of today in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

today = get\_today(timezone)

datetime\_naive = datetime.combine(today, end\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def first\_day\_of\_month(d):

"""

Given date return first day of the month in <type 'datetime.date'>

"""

return date(d.year, d.month, 1)

def first\_day\_of\_this\_month(timezone='UTC'):

"""

Return first day of this month in <type 'datetime.date'>.

"""

today = datetime.now(pytz.timezone(timezone)).date()

return date(today.year, today.month, 1)

def first\_day\_of\_this\_month\_in\_datetime(timezone='UTC'):

"""

Return first day of this month in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

first\_day = first\_day\_of\_this\_month(timezone)

datetime\_naive = datetime.combine(first\_day, start\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def last\_day\_of\_this\_month(timezone='UTC'):

"""

Return last day of this month in <type 'datetime.date'>

"""

today = datetime.now(pytz.timezone(timezone)).date()

return last\_day\_of\_month(today)

def last\_day\_of\_this\_month\_in\_datetime(timezone='UTC'):

"""

Return last day of this month in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

last\_day = last\_day\_of\_this\_month(timezone)

datetime\_naive = datetime.combine(last\_day, end\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def first\_day\_of\_last\_month(timezone='UTC'):

"""

Return first day of last month in <type 'datetime.date'>

"""

return first\_day\_of\_month(last\_day\_of\_last\_month(timezone))

def first\_day\_of\_last\_month\_in\_datetime(timezone='UTC'):

"""

Return first day of last month in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

first\_day = first\_day\_of\_last\_month(timezone)

datetime\_naive = datetime.combine(first\_day, start\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def last\_day\_of\_last\_month(timezone='UTC'):

"""

Return last day of last month in <type 'datetime.date'>

"""

last\_month\_date = first\_day\_of\_this\_month(timezone) - timedelta(days=1)

return last\_month\_date

def last\_day\_of\_last\_month\_in\_datetime(timezone='UTC'):

"""

Return last day of last month in <type 'datetime.datetime'>

Return is timezone aware datetime

"""

last\_day = last\_day\_of\_last\_month(timezone)

datetime\_naive = datetime.combine(last\_day, end\_time())

return add\_timezone\_to\_datetime\_naive(datetime\_naive, timezone)

def convert\_datetime\_to\_unix\_timestamp(datetime\_value):

"""

Takes datetime naive object and return value in unix timestamp seconds.

Return is float.

"""

epoch = datetime.utcfromtimestamp(0)

delta = datetime\_value - epoch

return delta.total\_seconds()

def convert\_datetime\_to\_unix\_timestamp\_millis(datetime\_value):

"""

Takes datetime naive object and return value in unix timestamp milliseconds.

Return is float.

"""

return convert\_datetime\_to\_unix\_timestamp(datetime\_value) \* 1000.0

def get\_current\_timestamp():

"""Get current timestamp, uses UTC

Returns:

int

"""

current\_time = datetime.now(pytz.timezone('UTC'))

return int(time.mktime(current\_time.timetuple()))

def get\_day\_of\_month():

return datetime.now(pytz.utc).day

def get\_day\_of\_week():

return datetime.now(pytz.utc).weekday()

def get\_hour\_of\_day():

return datetime.now(pytz.utc).hour

def time\_difference\_in\_ms(start, end):

"""Get time difference in milliseconds

Args:

start (datetime.datetime)

end (datetime.datetime)

Returns:

Integer time difference in milliseconds

"""

time\_difference = end - start

return int(time\_difference.total\_seconds() \* 1000)