Python Date and time

Python provides the **datetime** module work with real dates and times. In real-world applications, we need to work with the date and time. Python enables us to schedule our Python script to run at a particular timing.

In Python, the date is not a data type, but we can work with the date objects by importing the module named with **datetime, time, and calendar**.

In this section of the tutorial, we will discuss how to work with the date and time objects in Python.

The **datetime** classes are classified in the six main classes.

x

* **date** - It is a naive ideal date. It consists of the year, month, and day as attributes.
* **time** - It is a perfect time, assuming every day has precisely 24\*60\*60 seconds. It has hour, minute, second, microsecond, and **tzinfo** as attributes.
* **datetime** - It is a grouping of date and time, along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
* **timedelta -** It represents the difference between two dates, time or datetime instances to microsecond resolution.
* **tzinfo** - It provides time zone information objects.
* **timezone -** It is included in the new version of Python. It is the class that implements the **tzinfo** abstract base class.

Tick

In Python, the time instants are counted since 12 AM, 1st January 1970. The function **time()** of the module time returns the total number of ticks spent since 12 AM, 1st January 1970. A tick can be seen as the smallest unit to measure the time.

Consider the following example

**import** time;

1. #prints the number of ticks spent since 12 AM, 1st January 1970
2. **print**(time.time())

**Output:**

1585928913.6519969

How to get the current time?

The localtime() functions of the time module are used to get the current time tuple. Consider the following example.

**Example**

1. **import** time;
3. #returns a time tuple
5. **print**(time.localtime(time.time()))

**Output:**

time.struct\_time(tm\_year=2020, tm\_mon=4, tm\_mday=3, tm\_hour=21, tm\_min=21, tm\_sec=40, tm\_wday=4, tm\_yday=94, tm\_isdst=0)

Time tuple

The time is treated as the tuple of 9 numbers. Let's look at the members of the time tuple.

|  |  |  |
| --- | --- | --- |
| **Index** | **Attribute** | **Values** |
| 0 | Year | 4 digit (for example 2018) |
| 1 | Month | 1 to 12 |
| 2 | Day | 1 to 31 |
| 3 | Hour | 0 to 23 |
| 4 | Minute | 0 to 59 |
| 5 | Second | 0 to 60 |
| 6 | Day of weak | 0 to 6 |
| 7 | Day of year | 1 to 366 |
| 8 | Daylight savings | -1, 0, 1 , or -1 |

Getting formatted time

The time can be formatted by using the **asctime()** function of the time module. It returns the formatted time for the time tuple being passed.

**Example**

1. **import** time
2. #returns the formatted time
4. **print**(time.asctime(time.localtime(time.time())))

**Output:**

Tue Dec 18 15:31:39 2018

Python sleep time

The **sleep()** method of time module is used to stop the execution of the script for a given amount of time. The output will be delayed for the number of seconds provided as the float.

Consider the following example.

**Example**

1. **import** time
2. **for** i **in** range(0,5):
3. **print**(i)
4. #Each element will be printed after 1 second
5. time.sleep(1)

**Output:**

0

1

2

3

4

The datetime Module

The **datetime** module enables us to create the custom date objects, perform various operations on dates like the comparison, etc.

To work with dates as date objects, we have to import **the datetime** module into the python source code.

Consider the following example to get the **datetime** object representation for the current time.

**Example**

1. **import** datetime
2. #returns the current datetime object
3. **print**(datetime.datetime.now())

**Output:**

2020-04-04 13:18:35.252578

Creating date objects

We can create the date objects bypassing the desired date in the datetime constructor for which the date objects are to be created.

Consider the following example.

**Example**

1. **import** datetime
2. #returns the datetime object for the specified date
3. **print**(datetime.datetime(2020,04,04))

**Output:**

2020-04-04 00:00:00

We can also specify the time along with the date to create the datetime object. Consider the following example.

**Example**

1. **import** datetime
3. #returns the datetime object for the specified time
5. **print**(datetime.datetime(2020,4,4,1,26,40))

**Output:**

2020-04-04 01:26:40

In the above code, we have passed in **datetime()** function year, month, day, hour, minute, and millisecond attributes in a sequential manner.

Comparison of two dates

We can compare two dates by using the comparison operators like >, >=, <, and <=.

Consider the following example.

**Example**

1. **from** datetime **import** datetime as dt
2. #Compares the time. If the time is in between 8AM and 4PM, then it prints working hours otherwise it prints fun hours
3. **if** dt(dt.now().year,dt.now().month,dt.now().day,8)<dt.now()<dt(dt.now().year,dt.now().month,dt.now().day,16):
4. **print**("Working hours....")
5. **else**:
6. **print**("fun hours")

**Output:**

fun hours

The calendar module

Python provides a calendar object that contains various methods to work with the calendars.

Consider the following example to print the calendar for the last month of 2018.

**Example**

1. **import** calendar;
2. cal = calendar.month(2020,3)
3. #printing the calendar of December 2018
4. **print**(cal)

**Output:**

March 2020

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

Printing the calendar of whole year

The prcal() method of calendar module is used to print the calendar of the entire year. The year of which the calendar is to be printed must be passed into this method.

**Example**

1. **import** calendar
2. #printing the calendar of the year 2019
3. s = calendar.prcal(2020)

**Output:**