



## Unit -2

# **Working with Python Modules and Libraries**

# Content

- Installing and importing modules in Python
- Using built-in modules such as
  - math,
  - random,
  - datetime,
  - os,
  - functools,
  - itertools,
  - email,
  - urllib

# Installing and importing modules in Python



# What are Modules ?

- Consider a module to be the same as a code library.
- A module is simply a file containing Python code.
- This file can potentially include functions, variables, classes, constants, and executable code
- The filename becomes the module name.
- Python organizes its different functions into modules. When you run Python, it loads only a small set of functions from the built-in module. To use any other functions, you must import them.
- There are different Types of Modules in Python : inbuilt, external and user defined(customized)

# External Modules

- There are many other libraries that have been built by developers outside of the core Python team to add additional functionality to the language.
- These modules don't come as part of the Python language, but can be added in. **We call these external modules.**
- In order to use an external module, you must first install it on your machine.
- To install, you'll need to download the files from the internet to your computer, then integrate them with the main Python library so that the language knows where the module is located.
- You can find a list of popular modules here: [wiki.python.org/moin/UsefulModules](http://wiki.python.org/moin/UsefulModules)  
And a more complete list of pip-installable modules here: [pypi.org](http://pypi.org)

# Installing External Modules(1)

## Method 1 : Using pip

- The pip package manager is the best way to install Python 3 modules. However, modules that do not support pip can still be installed locally as long as they provide a setup.py file.
- Install the new Python module using the command:
  - **pip install <module-name>**
- The following example demonstrates how to install the ffmpeg-python module, which is used for media processing tasks.
  - **pip install ffmpeg-python**
- To list all installed Python modules and packages, use the pip list command.
  - **pip list**



# Installing External Modules(2)

## Method 1 : Using pip

- If you are using Jupyter Notebook or IPython Console, make sure to use ! before pip when you enter the command below. Otherwise it would return syntax error.
  - **!pip install package\_name**
- The ! prefix tells Python to run a shell command.
- **Syntax Error : Installing Package using PIP**
- Some users face error "SyntaxError: invalid syntax" in installing packages. To workaround this issue, run the command line below in command prompt -
  - **python -m pip install package-name**
- **Install Specific Versions of Python Package**
  - **python -m pip install Packagename==1.3** # specific version
  - **python -m pip install "Packagename>=1.3"** # version greater than or equal to 1.3

# Installing External Modules(3)

- **Method 2 : Using Command Prompt/Terminal**

1. Type "Command Prompt" in the search bar on Windows OS. For admin rights, right-click and choose "Run as administrator".
2. Search for folder named Scripts where pip applications are stored.
3. In command prompt, type `cd <file location of Scripts folder>` `cd` refers to change directory.
4. Type `pip install package-name`

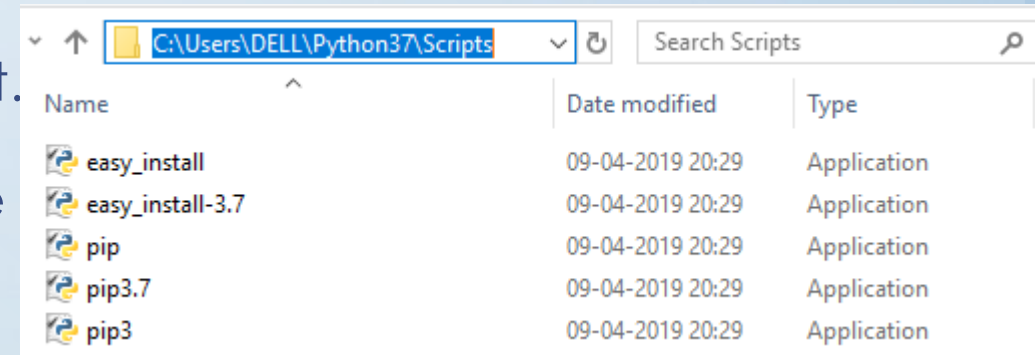
OR

```
Microsoft Windows [version 10.0.17134.648]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\DELL>cd C:\Users\DELL\Python37\Scripts

C:\Users\DELL\Python37\Scripts>pip install paramiko
Collecting paramiko
  Using cached https://files.pythonhosted.org/packages/cf/ae/94e70d49044ccc2
34bfbdb20114fa947d7ba6eb68a2e452d89b920e62227/paramiko-2.4.2-py2.py3-none-any.whl
Collecting cryptography>=1.5 (from paramiko)
  Using cached https://files.pythonhosted.org/packages/af/38/b53b461982d4c9fd9693beb7b6c6d77edd4fe7ceac7822aeb2a5754ccb15/cryptography-2.6.1-cp37-cp37m-win32.whl
Collecting pyasn1>=0.1.7 (from paramiko)
  Using cached https://files.pythonhosted.org/packages/7b/7c/c9386b82a25115cccf1903441bba3cbadcfae7b678a20167347fa8ded34c/pyasn1-0.4.5-py2.py3-none-any.whl
```

1. Search for folder named Scripts where pip applications are stored.
2. Select it and type `cmd`. It will open Command Prompt.
3. In Command Prompt , Type `pip install package-name`



Name	Date modified	Type
easy_install	09-04-2019 20:29	Application
easy_install-3.7	09-04-2019 20:29	Application
pip	09-04-2019 20:29	Application
pip3.7	09-04-2019 20:29	Application
pip3	09-04-2019 20:29	Application



# Other Useful Commands

- There are other useful commands using pip manager.

Description	Command
To uninstall a package	<code>pip uninstall package</code>
To upgrade a package	<code>pip install --upgrade package</code>
To search a package	<code>pip search "package-name"</code>
To check all the installed packages	<code>pip list</code>

# Load and Import Modules

- Once package is installed, next step is to load the package.
- There are several ways to load package or module in Python :
  1. import math loads the module math. Then you can use any function defined in math module using math.function.  
Refer the example below -

```
import math math.sqrt(4)
```

2. import math as m imports the math module under the alias m.

```
import math as m m.sqrt(4)
```

3. from math import \* loads the module math. Now we don't need to specify the module to use functions of this module.

```
from math import *  
sqrt(4)
```

4. imports the selected functions of the module math.

```
from math import sqrt, cos
```

# Load and Import Modules

- Once package is installed, next step is to load the package.
- There are several ways to load package or module in Python :
  1. import math loads the module math. Then you can use any function defined in math module using math.function.  
Refer the example below -

```
import math math.sqrt(4)
```

2. import math as m imports the math module under the alias m.

```
import math as m m.sqrt(4)
```

3. from math import \* loads the module math. Now we don't need to specify the module to use functions of this module.

```
from math import *  
sqrt(4)
```

4. imports the selected functions of the module math.

```
from math import sqrt, cos
```

# Using in-built Modules



# Math Module(1)

- Python has a built-in module that you can use for mathematical tasks.
- The math module has a set of methods and constants.

Constant	Description
<a href="#"><u>math.e</u></a>	Returns Euler's number (2.7182...)
<a href="#"><u>math.inf</u></a>	Returns a floating-point positive infinity
<a href="#"><u>math.nan</u></a>	Returns a floating-point NaN (Not a Number) value
<a href="#"><u>math.pi</u></a>	Returns PI (3.1415...)

- **Example:**

```
import math
print (math.e)
r = 4
pie = math.pi
print(pie * r * r)
```

# Math Module(2)

Method	Description
<a href="#"><code>math.ceil()</code></a>	Rounds a number up to the nearest integer
<a href="#"><code>math.cos()</code></a>	Returns the cosine of a number
<a href="#"><code>math.degrees()</code></a>	Converts an angle from radians to degrees
<a href="#"><code>math.exp()</code></a>	Returns E raised to the power of x
<a href="#"><code>math.fabs()</code></a>	Returns the absolute value of a number
<a href="#"><code>math.factorial()</code></a>	Returns the factorial of a number
<a href="#"><code>math.floor()</code></a>	Rounds a number down to the nearest integer
<a href="#"><code>math.fmod()</code></a>	Returns the remainder of x/y
<a href="#"><code>math.fsum()</code></a>	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
<a href="#"><code>math.gcd()</code></a>	Returns the greatest common divisor of two integers
<a href="#"><code>math.isnan()</code></a>	Checks whether a value is NaN (not a number) or not
<a href="#"><code>math.isqrt()</code></a>	Rounds a square root number downwards to the nearest integer
<a href="#"><code>math.log()</code></a>	Returns the natural logarithm of a number, or the logarithm of number to base
<a href="#"><code>math.log10()</code></a> / <a href="#"><code>math.log2()</code></a>	Returns the base-10 logarithm / base-2 logarithm of x
<a href="#"><code>math.pow()</code></a>	Returns the value of x to the power of y
<a href="#"><code>math.prod()</code></a>	Returns the product of all the elements in an iterable
<a href="#"><code>math.radians()</code></a>	Converts a degree value into radians
<a href="#"><code>math.sin()</code></a>	Returns the sine of a number
<a href="#"><code>math.sqrt()</code></a>	Returns the square root of a number
<a href="#"><code>math.tan()</code></a>	Returns the tangent of a number



# Math Module(3)

- Example:

```
import math
```

```
a = 2.3
```

```
print (math.ceil(a)) #Ceiling – round to largest value
```

```
print (math.floor(a)) #Floor – round to smallest value
```

```
a = 5
```

```
print(math.factorial(a)) #Factorial
```

```
print(math.sqrt(4)) #Square root
```

```
a = math.pi/6
```

```
print (math.sin(a)) #sin
```

```
print (math.cos(a))
```

```
print (math.tan(a))
```

# Random Module(1)

- Python has a built-in module that you can use to make random numbers.
- The random module has a set of methods:

Method	Description
<a href="#"><code>randrange()</code></a>	Returns a random number between the given range
<a href="#"><code>randint()</code></a>	Returns a random number between the given range
<a href="#"><code>choice()</code></a>	Returns a random element from the given sequence
<a href="#"><code>choices()</code></a>	Returns a list with a random selection from the given sequence
<a href="#"><code>shuffle()</code></a>	Takes a sequence and returns the sequence in a random order
<a href="#"><code>sample()</code></a>	Returns a given sample of a sequence
<a href="#"><code>random()</code></a>	Returns a random float number between 0 and 1
<a href="#"><code>uniform()</code></a>	Returns a random float number between two given parameters

# Random Module(2)

- Example:

```
import random
```

```
print(random.random()) # random float number between 0.0 to 1.0.
```

```
print(random.randint(1, 100)) # random integer between specified integers
```

```
print(random.randrange(1, 10,2)) # random integer between specified  
range
```

```
print(random.choice([12, 23, 45, 78, 95])) # random selected element from  
sequence
```

```
numbers=[12,23,45,67,65,43]
```

```
random.shuffle(numbers) #randomly reorders the elements
```

```
print(numbers)
```

# Random Module(1)

- Python has a built-in module that you can use to make random numbers.
- The random module has a set of methods:

Method	Description
<a href="#"><code>randrange()</code></a>	Returns a random number between the given range
<a href="#"><code>randint()</code></a>	Returns a random number between the given range
<a href="#"><code>choice()</code></a>	Returns a random element from the given sequence
<a href="#"><code>choices()</code></a>	Returns a list with a random selection from the given sequence
<a href="#"><code>shuffle()</code></a>	Takes a sequence and returns the sequence in a random order
<a href="#"><code>sample()</code></a>	Returns a given sample of a sequence
<a href="#"><code>random()</code></a>	Returns a random float number between 0 and 1
<a href="#"><code>uniform()</code></a>	Returns a random float number between two given parameters

# DateTime Module(1)

- Python Datetime module supplies classes to work with date and time. These classes provide several functions to deal with dates, times, and time intervals.
- Date and DateTime are an object in Python, so when you manipulate them, you are manipulating objects and not strings or timestamps.
- **Main Classes of the datetime Module**
- There are mainly 6 classes of datetime module present, which are listed below:
  1. **date class:** It represents a date (year, month, and day).
  2. **time class:** It represents a time of day (hours, minutes, seconds, and microseconds).
  3. **datetime class:** It represents a combination of a date and a time and provides a range of methods to manipulate and work with date-time values.
  4. **timedelta class:** It represents a duration of time (days, seconds, and microseconds) and can be used to perform arithmetic operations on date-time values.
  5. **tzinfo class:** It can be used to define time zones for date-time objects. It is an abstract base class.
  6. **timezone class:** The timezone class is a subclass of tzinfo class in the datetime module that represents a fixed offset from UTC (Coordinated Universal Time).

# Date Class(1)

- The date class is used to instantiate date objects in Python. When an object of this class is instantiated, it represents a date in the format YYYY-MM-DD.
- The constructor of this class needs three mandatory arguments year, month, and date.
- **Syntax: class datetime.date(year, month, day)**
- The arguments must be in the following range –
  - MINYEAR <= year <= MAXYEAR
  - 1 <= month <= 12
  - 1 <= day <= number of days in the given month and year

- **Example:**

## 1. Create a data object

```
from datetime import date  
date1 = date(2021, 2, 10)  
print("Date is :", date1)
```



# Date Class(2)

- **Example:**

## 2. Get Today's Date

```
from datetime import date
```

```
today = date.today() # Get today's date
```

```
print('Today:', today)
```

```
# extract attributes
```

```
print("Year:", today.year)
```

```
print("Month:", today.month)
```

```
print("Day:", today.day)
```

```
date_string = today.strftime("%d/%m/%Y") # Format a date object as a string
```

```
print(date_string)
```

# Date Class(3)

- List of Date Class Methods:

Function Name	Description
ctime()	Return a string representing the date
fromtimestamp()	Returns a date object from the POSIX timestamp
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Returns the string representation of the date
isoweekday()	Returns the day of the week as an integer where Monday is 1 and Sunday is 7
replace()	Changes the value of the date object with the given parameter
strftime()	Returns a string representation of the date with the given format
today()	Returns the current local date
weekday()	Returns the day of the week as integer where Monday is 0 and Sunday is 6

# Time Class(1)

- The time class creates the time object which represents local time, independent of any day.
- **Syntax: `class datetime.time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None, *, fold=0)`**

All the arguments are optional. tzinfo can be None otherwise all the attributes must be integer in the following range –

$0 \leq \text{hour} < 24$

$0 \leq \text{minute} < 60$

$0 \leq \text{second} < 60$

$0 \leq \text{microsecond} < 1000000$

fold in [0, 1]

- **Example:**

## 1. Create a time object

```
from datetime import time
```

```
time1 = time(12, 14, 36)
```

```
print("Time is :", time1)
```

# Time Class(2)

Attribute Name	Description
min	Minimum possible representation of time
max	Maximum possible representation of time
resolution	The minimum possible difference between time objects
hour	The range of hour must be between 0 and 24 (not including 24)
minute	The range of minute must be between 0 and 60 (not including 60)
second	The range of second must be between 0 and 60 (not including 60)
microsecond	The range of microsecond must be between 0 and 1000000 (not including 1000000)
tzinfo	The object containing timezone information
fold	Represents if the fold has occurred in the time or not

## Example:

```
from datetime import time
print("Min Time Supported:", time.min)
print("Max Time Supported:", time.max)
Time= time(12,24,36,1212)
print("Hour:",Time.hour)
print("Minute:",Time.minute)
print("Second:",Time.second)
print("MicroSecond:",Time.microsecond)
```

# Time Class(3)

Function Name	Description
fromisoformat()	Returns a time object from the string representation of the time
isoformat()	Returns the string representation of time from the time object
replace()	Changes the value of the time object with the given parameter
strftime()	Returns a string representation of the time with the given format

## Example:

```
from datetime import time
```

```
Time = time(12,24,36,1212) # Creating Time object
```

```
Str = Time.isoformat() # Converting Time object to string
```

```
print("String Representation:", Str)
```

```
Time = "12:24:36.001212"
```

```
Time = time.fromisoformat(Str) # Converting string to Time
```

```
print("\nTime from String", Time) from datetime import time
```

```
Time=Time.replace(hour = 13,second = 12)# Replace hour
```

```
print("New Time:", Time)
```

```
Ftime = Time.strftime("%I:%M %p") # Formatting Time
```

```
print("Formatted time", Ftime)
```

```
now = datetime.now() # Extract time from datetime object
```

```
print("Current time is:", now.time())
```

# DateTime Class(1)

- The datetime class contains information about both date and time. It Assumes the Gregorian Calendar and there are  $360 \times 24$  seconds every day.
- **Syntax: `datetime.datetime(year, month, day, hour, minute=0, second, microsecond, tzinfo=None)`**
- Arguments must be integers in the following ranges:
  - `MINYEAR <= year <= MAXYEAR`,
  - `1 <= month <= 12`,
  - `1 <= day <= number of days in the given month and year`,
  - `0 <= hour < 24`,
  - `0 <= minute < 60`,
  - `0 <= second < 60`,
  - `0 <= microsecond < 1000000`,
  - fold in `[0, 1]`.

- **Example:**

```
# import datetime class
from datetime import datetime

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

print('Current datetime:', now)

dt = datetime(year=2021, month=2, day=17, hour=13, minute=47, second=34) # create datetime object

print("Datetime is:", dt)
```



# DateTime Class(2)

Attribute Name	Description
min	The minimum representable DateTime
max	The maximum representable DateTime
resolution	The minimum possible difference between datetime objects
year	The range of year must be between MINYEAR and MAXYEAR
month	The range of month must be between 1 and 12
day	The range of days must be between 1 and the number of days in the given month of the given year
hour	The range of hours must be between 0 and 24 (not including 24)
minute	The range of minutes must be between 0 and 60 (not including 60)
second	The range of second must be between 0 and 60 (not including 60)
microsecond	The range of microseconds must be between 0 and 1000000 (not including 1000000)
tzinfo	The object containing timezone information
fold	Represents if the fold has occurred in the time or not

- Example:**

```
from datetime import datetime
print("Min DateTime:",datetime.min)
print("Max DateTime:",datetime.max)
```

## # Getting Today's Datetime

```
today = datetime.now()
print("Day: ", today.day)
print("Month: ", today.month)
print("Year: ", today.year)
print("Hour: ", today.hour)
print("Minute: ", today.minute)
print("Second: ", today.second)
```

# DateTime Class(3)

Function Name	Description
combine()	Combines the date and time objects and returns a DateTime object
ctime()	Returns a string representation of the date and time
date()	Return the Date class object
fromisoformat()	Returns a datetime object from string representation of date and time
isocalendar()	Returns a tuple year, week, and weekday
isoformat()	Return the string representation of the date and time
isoweekday()	Returns the day of the week as integer where Monday is 1 & Sunday is 7
now()	Returns current local date and time with tz parameter
replace()	Changes the specific attributes of the DateTime object
today()	Return local DateTime with tzinfo as None
weekday()	Returns the day of the week as integer where Monday is 0 & Sunday is 6

- Example:**

```
from datetime import datetime
```

## # Getting Today's Datetime

```
today = datetime.now()
```

```
print("Today's date using now()  
method:", today)
```

```
today = datetime.today()
```

```
print("Today's date using today()  
method:", today)
```

```
from datetime import datetime
```

```
# current dateTime
```

```
now = datetime.now()
```

```
# convert to string
```

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

```
print('DateTime String:',  
date_time_str)
```

# Timedelta Class(1)

- Timedelta class is used for calculating differences between dates and represents a duration. The difference can both be positive as well as negative.
- **Syntax: class datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)**

Attribute Name	Description
min	minimum value of timedelta object is -999999999
max	maximum value of timedelta object is 999999999
resolution	The minimum possible difference between timedelta objects

Method Name	Description
total_seconds()	Returns the duration represented by the timedelta object as number of seconds.

Operator	Description
Addition (+)	Adds and returns two timedelta objects
Subtraction (-)	Subtracts and returns two timedelta objects
Multiplication (*)	Multiplies timedelta object with float or int
Division (/)	Divides the timedelta object with float or int
Floor division (//)	Divides the timedelta object with float or int and return the int of floor value of the output
Modulo (%)	Divides two timedelta object and returns the remainder
+(timedelta)	Returns the same timedelta object
-(timedelta)	Returns the resultant of -1*timedelta
abs(timedelta)	Returns the +(timedelta) if timedelta.days > 1=0 else returns -(timedelta)
str(timedelta)	Returns a string in the form (+/-) day[s], HH:MM:SS.UUUUUU
repr(timedelta)	Returns the string representation in the form of the constructor call

# Timedelta Class(1)

- **# Timedelta function demonstration**

```
from datetime import datetime, timedelta  
  
# creating datetime objects  
date1 = datetime(2020, 1, 3)  
date2 = datetime(2020, 2, 3)  
diff = date2 - date1  # difference between dates  
print("Difference in dates:", diff)  
  
date1 += timedelta(days = 4) # Adding days to date1  
print("Date1 after 4 days:", date1)  
  
date1 -= timedelta(15) # Subtracting days from date1  
print("Date1 before 15 days:", date1)  
  
t1=timedelta(days=1) #timedelta object  
print("String representation:", str(t1)) # Getting string representation  
print("Constructor call:", repr(t1)) # Getting Constructor call
```

# Calendar Class

- The Calendar module in Python is used to display calendars and provides useful Built-in functions for displaying week, week day, month, month of the year, and other operations.
- By default, these calendars have Monday as the first day of the week, and Sunday as the last.
- **Example:**

```
import calendar
```

```
print(calendar.calendar(2022)) #display the calender
```

```
print(calendar.month(2022,8)) #display august month calender
```

```
print(f'Monday Weekday Number: {calendar.MONDAY}') #get weekday number
```

```
print(calendar.isleap(2020)) #check is it leap year
```

# os module(1)

- Python has a built-in os module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.

Sr.No.	Methods with Description
1	<a href="#"><u>os.chdir(path)</u></a> Change the current working directory to path
2	<a href="#"><u>os.chroot(path)</u></a> Change the root directory of the current process to path.
3	<a href="#"><u>os.close(fd)</u></a> Close file descriptor fd.
4	<a href="#"><u>os.getcwd()</u></a> Return a string representing the current working directory.
5	<a href="#"><u>os.listdir(path)</u></a> Return a list containing the names of the entries in the directory given by path.
6	<a href="#"><u>os.makedirs(path[, mode])</u></a> Recursive directory creation function.
7	<a href="#"><u>os.mkdir(path[, mode])</u></a> Create a directory named path with numeric mode.
8	<a href="#"><u>os.open(file, flags[, mode])</u></a> Open the file and set various flags and possibly its mode
9	<a href="#"><u>os.read(fd, n)</u></a> Read at most n bytes from file descriptor fd. Return a string containing the bytes read.
10	<a href="#"><u>os.remove(path)</u></a> Remove the file path.
11	<a href="#"><u>os.removedirs(path)</u></a> Remove directories recursively.
12	<a href="#"><u>os.rename(src, dst)</u></a> Rename the file or directory src to dst.
13	<a href="#"><u>os.rmdir(path)</u></a> Remove the directory path
14	<a href="#"><u>os.write(fd, str)</u></a> Write the string str to file descriptor fd. Return the number of bytes actually written.



# os module(2)

- The `os.path` module is a very extensively used module that is handy when processing files from different places in the system. It is used for different purposes such as for merging, normalizing and retrieving path names in python.
- some OS Path functions by which we can do Path Manipulation in the Python OS Module:
- **`os.path.basename(path)`**: This function gives us the last part of the path which may be a folder or a file name.

**Ex:**

```
import os
file = os.path.basename("C:\Users\xyz\Documents\My Web Sites\intro.html")
print(file)
```

- **`os.path.dirname(path)`**: This function gives us the directory name where the folder or file is located.

**Ex:**

```
DIR = os.path.dirname("C:\Users\xyz\Documents\My Web Sites")
print(DIR)
```

- **`os.path.isdir(path)`**: This function specifies whether the path is existing directory or not.

**Ex:**

```
Out=os.path.isdir("C:\\Users")
print(Out)
```

- **`os.path.isfile(path)`**: This function specifies whether the path is existing file or not.

**Ex:**

```
Out=os.path.isfile("C:\\Users\demo.py")
print(Out)
```

# functools module(1)

- The functools module is part of Python's standard library and was implemented for higher-order functions (functions that either accept a function as an input or return another function as an output. For example, map(), filter(), etc.)
- It is used in functional programming, it provides various tools to use functions, decorators, etc.
- There are two classes of functools module.

1. **Partial Class:** This allows to create new function with some of original function's arguments that are already set. This can be helpful when you want to simplify function calls by fixing specific arguments.

**Syntax:** partial(function, /, \*args, \*\*keywords)

**Example:** from functools import \*

```
def power(a,b):
```

```
    return a**b
```

```
pow=partial(power,b=4) #partial class
```

```
print(pow(2)) #partial class object
```

2. **Partialmethod class:** It is like partial class, but it is designed specifically for methods within classes.

**Ex:** class Demo:

```
    self.color = 'black'
```

```
    def _color(self, type):
```

```
        self.color = type
```

```
    set_red = partialmethod(_color, type='red')
```

```
obj = Demo()
```

```
obj.set_red()
```

# functools module(2)

- **Functions /Methods in functools Module:**

1. **reduce(function, iterable)** : It applies a function of two arguments repeatedly on the elements of a sequence so as to reduce the sequence to a single value.

```
from functools import *
```

```
list1=[2, 4, 6, 9, 1, 3]
```

```
sum=reduce(lambda a, b: a+b, list1)
```

2. **Total Ordering:** It is a class decorator that fills in the missing comparison methods (`__lt__`, `__gt__`, `__eq__`, `__le__`, `__ge__`). If a class is given which defines one or more comparison methods, “@total\_ordering” automatically supplies rest as per given definitions. However, the class must define one of `__lt__()`, `__le__()`, `__gt__()`, or `__ge__()` and additionally, class should supply an `__eq__()` method.

## @total\_ordering

```
class Student:
```

```
    def __init__(self, name, standard, age, percentage):
```

```
        self.name = name
```

```
        self.percentage = percentage
```

```
    def __eq__(self, anotherS):
```

```
        return self.percentage == anotherS.percentage
```

```
    def __lt__(self, anotherS):
```

```
        return self.percentage < anotherS.percentage
```

```
print(Student("Jack", 88) < Student("Marc", 84))
```

```
print(Student("Cumin", 65) > Student("Michek", 80))
```

```
print(Student("Daisy", 78) == Student("Rose", 78))
```

# functools module(3)

- **Functions /Methods in functools Module:**

**3. lru\_cache(maxsize,typed):** stands for **Least-Recently-Used Cache** is decorator which uses **Memoization** is caching technique that assures that a function does not have to be run multiple times for same inputs. Instead, function's output is saved in memory first time it is called and can be retrieved later if necessary.

- **Example:**

**@lru\_cache(maxsize = None)**

```
def factorial(n):  
    if n<= 1:  
        return 1  
    return n * factorial(n-1)  
print([factorial(n) for n in range(7)])  
print(factorial.cache_info())
```

**4. singledispatch():** function decorator which transforms function into generic function. It can have different behaviors depending upon type of its first argument. It is used for function overloading which are registered using register() attribute.

**Example:**

**@singledispatch**

```
def fun(s):  
    print(s)  
@fun.register(int)  
def _(s):  
    print(s * 2)  
fun('python')  
fun(10)
```

# functools module(4)

5. **wraps()**: The wraps() function is a decorator. When we use wraps decorator, it ensures that when we create new decorator, it doesn't lose important properties of the original function.
- In simple words, it acts like a wrapper around the new decorator. This makes sure that it looks and behaves like original function.

- **Example:**

```
def decorator(f):
```

```
    @wraps(f)
```

```
    def decorated(*args, **kwargs):
```

```
        """Decorator's docstring"""
```

```
        return f(*args, **kwargs)
```

```
    print('Documentation of decorated :', decorated.__doc__)
```

```
    return decorated
```

```
@decorator
```

```
def f(x):
```

```
    """f's Docstring"""
```

```
    return x
```

```
print('f name :', f.__name__)
```

```
print('Documentation of f :', f.__doc__)
```

# functools module(5)

6. **Cmp\_to\_key (iterable, key=cmp\_to\_key(cmp\_function))** :It converts a comparison function into a key function. The comparison function **must return 1, -1 and 0** for different conditions.
- It can be used in key functions such as **sorted(), min(), max()**.

**# function to sort according to last character**

```
def cmp_fun(a, b):  
    if a[-1] > b[-1]:  
        return 1  
    elif a[-1] < b[-1]:  
        return -1  
    else:  
        return 0
```

```
list1 = ['geeks', 'for', 'geeks']  
l = sorted(list1, key = cmp_to_key(cmp_fun))  
print('sorted list :', l)
```



# itertools module(1)

- Itertools is a module in python, it is used to iterate over data structures that can be stepped over using a for-loop. Such data structures are also known as iterables.
- This module incorporates functions that utilize computational resources efficiently. Using this module also tends to enhance the readability and maintainability of the code.
- The itertools module needs to be imported prior to using it in the code.
- Different types of iterators provided by this module are:
- **Infinite iterators:** In Python, any object that can implement for loop is called iterators. Lists, tuples, set, dictionaries, strings are the example of iterators but iterator can also be infinite and this type of iterator is called infinite iterator.

Iterator	Argument	Description
count(start,step)	start, [step]	It prints from the start value to infinite.
cycle()	P	This iterator prints all value in sequence from the passed argument in cyclic manner.
repeat()	elem [,n]	it repeatedly prints the passed value for infinite time.



# itertools module(2)

**#count() – start from 10 with step=5**

```
import itertools
for i in itertools.count(10,5):
    if i == 50:
        break
    else:
        print(i,end=" ")
```

**#cycle() – repeats 1,2,3 for 7 times**

```
import itertools
temp = 0
for i in itertools.cycle("123"):
    if temp > 7:
        break
    else:
        print(i,end=' ')
        temp = temp+1
```

**# repeat() -print hello two times**

```
import itertools
result = itertools.repeat('hello',
times = 2)
for word in result:
    print (word)
```

- **Combinatoric iterators:** The complex combinatorial constructs are simplified by recursive generators. The permutations, combinations, and Cartesian products are example of combinatoric construct.

Iterator	Argument	Description
<b>Product()</b>	Iterable ,repeat	It is used to calculate the cartesian product of input iterable.
<b>permutations()</b>	Iterable, groupsize	generate all possible permutations of an iterable. Every element is decided to be unique based on its position and not its value.
<b>Combinations()</b>	Iterable, groupsize	generate all possible unique combinations for iterable with specified group_size in sorted order.
<b>Combinations_with_replacement()</b>	Iterable, groupsize	similar to combinations(), but It allows individual elements in iterable to get repeated more than once.

# itertools module(3)

## #product()

```
from itertools import product
print(list(product([1, 2], repeat=2)))
#O/P: [(1, 1), (1, 2), (2, 1), (2, 2)]
```

## #permutations()

```
from itertools import permutations
print(list(permutation('AB')
#O/P: [('A', 'B'), ('B', 'A')]
```

## # combinations()

```
from itertools import *
print(list(combinations('ABC',2)
#O/P:[('A', 'B'), ('A', 'C'), ('B', 'C')]
```

## # combinations()

```
from itertools import *
print(list(combinations_with_replacement('ABC',2)
#O/P[('A', 'A'), ('A', 'B'), ('A', 'C'), ('B', 'B'), ('B', 'C'), ('C', 'C')]
```

- **Terminating iterators:** Terminating iterators are generally used to work on the small input sequence and generate the output based on the functionality of the method used in iterator.
- **For Functions of Terminating Iterators**  
<https://www.scaler.com/topics/itertools-in-python/>

# email module(1)

- Python provides a couple of really nice modules that we can use to craft emails with. They are the **email** and **smtplib** modules.
- An application that handles and delivers e-mail over the Internet is called a "mail server" -**Simple Mail Transfer Protocol (SMTP)**.
- Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mails to any Internet machine.
- To send an email, you need to obtain the object of SMTP class with the following function –

```
import smtplib  
smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

- The SMTP object has following methods –
- **login(user, password)** – Log in on an SMTP server that requires authentication.
- **quit()** – terminate the SMTP session.
- **ehlo(name)** – Hostname to identify itself.
- **starttls()** – puts the connection to the SMTP server into TLS mode.
- **sendmail( from\_addr, to\_addrs,message)** – converts message to a bytestring and passes it to sendmail.

# email module(2)

- **Example: gmail smtp**

```
import smtplib
mail=smtplib.SMTP('smtp.gmail.com', 587)
mail.ehlo()#introduce the connection to itself
mail.starttls() #encrypts SMTP Connection
sender='mvl@gmail.com'
recipient='tester@gmail.com'
mail.login(sender,'*****') #login using credentials
message = """\
Subject: My First Email
Hello there, this is my first email sent using Python. """
mail.sendmail(sender, recipient, message)
mail.close() #close the connection
```

# urllib module(1)

- The Python urllib module allows us to access the website via Python code.
- It is a collection of various modules that helps us to work with the URLs (Uniform Resource Locator).
- We can make various types of HTTP web requests like GET, POST, PUT, and DELETE using the urllib Python library. We can also deal with JSON data in our Python code using this library.
- Some of the use cases of the urllib Python library are:
  - We can download data.
  - We can access websites.
  - We can perform web scraping.
  - We can parse data.
  - We can modify webpage headers.
- The urllib module consists of several modules for working with the URLs. These are given below –
  1. The **urllib.request** for opening and reading.
  2. The **urllib.parse** for parsing URLs
  3. The **urllib.error** for the exceptions raised
  4. The **urllib.robotparser** for parsing robot.txt files

# urllib module(2)

## 1. urllib.request

- We most often use urllib.request to open and read data, or the source code of the page.
- We generally use the urlopen(url) function of the urllib.request function to open webpages.
- We need to import the urllib module and assign the opening of the URL to a variable, where we can use the read() command to read the data.
- **Example:**

```
from urllib.request import urlopen
```

```
url = urlopen("https://www.python.org/")
```

```
print(url.read()) #display header request
```

```
print(url.info())#provides a dictionary of header request
```

```
print ('Content Type = ', url.info()["content type"])
```



# urllib module(3)

## 2 . urllib.parse

- The urllib.parse helps to define the function to manipulate URLs and their components parts to create or destroy them.
- We can parse the URL to check if it is a valid one or not
- The prime use of the urllib.parse module is used to split the URL into several smaller components. We can also use this module to join the various components of the URL to make the URL string.

Function Name	Use
urllib.parse.urlparse	separate the various components of a URL.
urllib.parse.urlunparse	join back the various components of a URL.
urllib.parse.urlsplit	behaves similar to the urllib.parse.urlparse function but it does not split the components of the URL.
urllib.parse.urlunsplit	combine the components of the URL (in the form of tuple) returned by the urllib.parse.urlsplit function.
urllib.parse.urldefrag	remove fragments from a URL if there are fragment(s) present in the URL.



# urllib module(4)

## 2 . urllib.parse

- **Example:**

```
from urllib.parse import *  
url = urlparse ('https://www.python.org/')#getting various parts of the URL  
print (url)
```

---

```
unparse=urlunparse(url) #generate the original URL string  
print(unparse)
```

---

```
url = 'https://www.google.com/search'  
values = {'query' : 'python tutorial'}  
data = urlencode(values)  
data = data.encode('utf-8')# data should be bytes  
print(data)
```

---

```
url='https://google.com/python'  
value = urllib.parse.urlsplit(url)#split the url into parts  
print(value)
```

# urllib module(5)

## 3. urllib.error

- This module defines the classes for exception raised by urllib.request. Whenever there is an error in fetching a URL, this module helps in raising exceptions.
- The following are the exceptions raised :
  - **URLError** – It is raised for the errors in URLs, or errors while fetching the URL due to connectivity, and has a 'reason' property that tells a user the reason of error.
  - **HTTPError** – It is raised for the exotic HTTP errors, such as the authentication request errors. It is a subclass of URLError. Typical errors include '404' (page not found), '403' (request forbidden), and '401' (authentication required).

```
# URL Error : # trying to read the URL but with no internet connectivity
import urllib.request
import urllib.parse
try:
    x =
urllib.request.urlopen('https://www.google.com')
    print(x.read())
except Exception as e : # Catching exception generated
    print(str(e))
```

```
# HTTP Error :# trying to read the URL
import urllib.request
import urllib.parse
try:
    x =
urllib.request.urlopen('https://www.google.com / search?q =
test')
    print(x.read())
except Exception as e : # Catching the exception generated
    print(str(e))
```

# urllib module(6)

## 4 . urllib.parse

- This module contains a single class, RobotFileParser. This class answers question about whether or not a particular user can fetch a URL that published robot.txt files.
- Robots.txt is a text file webmasters create to instruct web robots how to crawl pages on their website. The robot.txt file tells the web scraper about what parts of the server should not be accessed.

- **Example:**

```
import urllib.robotparser as rb
```

```
bot = rb.RobotFileParser()
```

```
# checks where the website's robot.txt file reside
```

```
robot = bot.set_url('https://www.google.com / robot.txt')
```

```
print(robot)
```

```
file = bot.read() # reads the files
```

```
print(file)
```

```
page = bot.can_fetch('*', 'https://www.geeksforgeeks.org/') # we can crawl the main site
```

```
print(page)
```