Python Modules

Modules: Creating modules, import statement, from. Import statement, name spacing, builtin modules- os, random, math, cmath, pprint, json, request, date, RegEx.

3.0 What is a Module?

- Python module can be defined as a python program file which contains a python code including python statements, variables, functions and classes.
- Python code file saved with the extension (.py) is treated as the module. runnable code inside the python module.
- The module name is the filename, i.e., a module shall be saved as "<module_name>.py".
- A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use.
- Module defines our most used functions in a module and imports it, modules to break down large programs into small manageable and organized files.
- Python has lot of built-in modules; math module is one of them. math module provides most of the familiar mathematical functions.
- To use the functionality of one module into another, we must have to import the specific module.

  Syntax: import <module-name>
- Every module has its own functions, those can be accessed with . (dot)
- Before we can use the functions in a module, we have to import it with an import statement:

```
>>> import math
```
- This statement import a module object named math. If you display the module object, you get some information about it:

```
>>> math
<module 'math' (built-in)>
```

Advantages of modules

- Reusability: Working with modules makes the code reusability a reality.
- Simplicity: Module focuses on a small proportion of the problem,rather than focusing on the entire problem.
- Scoping:A separate namespace is defined by a module that helps to avoid collisions between identifiers.

3.1 Create A Module :

- In Python, a module is a self-contained Python file that contains Python statements and definitions
- To create a module just save the code you want in a file with the file extension .py:

- It shall be saved with file extension of ".py". The module name is the filename, i.e., a module shall be saved as "<module_name>.py".
- A module is simply a Python file with a .py extension that can be imported inside another Python program.
- The name of the Python file becomes the module name.
- The module contains definitions and implementation of classes, variables, and functions that can be used inside another program.
- A module typically begins with a triple-double-quoted documentation string (doc-string) (available in <module_name>.__doc__), followed by variable, function and class definitions.

Example: The greet Module

Create a module called greet (python file:save as "greet.py") as follows:

#source code: create python file with the name greet.py

```
msg = 'Hello'         # Global Variable msg
def greet(name):    # Function name  greet()
print('{}, {}'.format(msg, name))
```

The import statement:
after creating the module To use an external module in your script, use the import statement:

```
import <module_name>        # import one module

>>> import greet
>>> greet.mgret('Madhu')     # <module_name>.<function_name>(positional argument)
Hai ! Hello!! , Madhu
>>> print(greet.msg)            # <module_name>.<var_name>
Hai ! Hello!!

>>> greet.__doc__               # module's doc-string
' the greet module with attributes msg and mgret()'

>>> greet.__name__              # module's name
'greet'

>>> dir(greet)                  # List all attributes defined in the module
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__',
'__spec__', 'mgret', 'msg']

>>> help(greet)                 # Show module's name, functions, data, ...
Help on module greet:
NAME
    greet - the greet module with attributes msg and greet()
FUNCTIONS
    mgret(name)
DATA
    msg = 'Hai ! Hello!! '
FILE
```

c:\users\mohith\desktop\greet.py

****

3.2 How to import modules in Python? or Use a Module

- import the definitions inside a module to another module or the interactive interpreter in Python. use the import keyword to do this.
- Using the module name we can access the function using the dot (**.**) operator.
- To import our previously defined module, we type the following in the Python prompt.

There are various ways to import modules. They are listed below..

- Python import statement
- Import with renaming
- Python from...import statement
- Import all names (*)

The dir() built-in function:It shows all built-in and user-defined modules.

Python import statement

- import a module using the import statement and access the definitions inside it

  using the dot operator (**.**).
- To use an external module in your script, use the import statement:
- Syntax

```
import <module_name>                    # import one module
<module_name>.<function_name>           # Access the function in the module
```

- Here is an example.

```
# import statement example
# To import standard module math

import math
print("The value of pi is", math.pi)
```

When you run the program, the output will be:

```
The value of pi is 3.141592653589793
```

Import with renaming

We can import a module by renaming it as follows:

# import module by renaming it

syntax:

```
<import > <module name> as <rename or newname >     # To reference the imported module as <name>

>>> import math as m

>>> print("The value of pi is", m.pi)
```

We have renamed the math module as m.This can save us typing time in some cases.

Note: that the name math is not recognized in our scope. Hence, math.pi is invalid, and m.pi is the correct implementation.

Python from...import statement

We can import specific names from a module without importing the module as a whole.

With the from-import statement, you can reference the imported attributes using <attr_name> directly, without qualifying with the <module_name>.

Here is an example.

syntax:

```
from <module_name> import <attr_name>                    # import one attribute
from <module_name> import <attr_name_1>, <attr_name_2>, ... # import selected attributes
```

or

```
from <module_name > import <fun_name_1>,<fun_2>,<fun_3>,...<fun_n>
```

Example: from... import

```
# import only pi from math module
>>>from math import pi
    print("The value of pi is", pi)
```

Here, we imported only the pi attribute from the math module.In such cases, we don't use the dot operator.

We can also import multiple attributes as follows:

```
>>> from math import pi, e
>>> pi
3.141592653589793
>>> e
2.718281828459045
```

Import all names

We can import all names (definitions) from a module using the following construct:

Syntax:

```
from <module_name> import *          # import ALL attributes
# import all names from the standard module math
>>> from math import *
>>> print("The value of pi is", pi)
```

Here, we have imported all the definitions from the math module. This includes all names visible in our scope except those beginning with an underscore(private definitions).

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

```
# For example, mycal.py (program to simple calculator)
def add(a,b):
    print(a+b)
def sub(a,b):
    print(a-b)
def mul(a,b):
    print(a*b)
def div(a,b):
    print(a/b)
#call the module (mycal)
>>>from mycal import*
>>> a=int(input('enter value of a'))
enter value of a 23
>>> b=int(input('enter value of b'))
enter value of b 34
>>> add(a,b)
57
>>> mul(a,b)
782
>>> sub(a,b)
-11
>>> div(256,8)
32.0
```

****

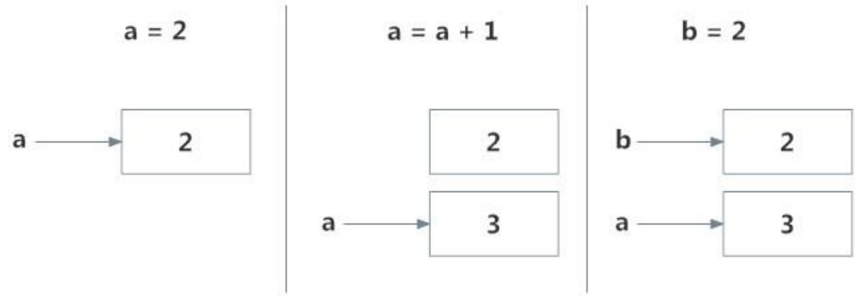
3.3 Namespace: **Names, Namespaces and Scope**

Names : What are names in Python?

- In Python, a name is roughly analogous to a variable in other languages but with some extras.
- Because of the dynamic nature of Python, a name is applicable to almost everything, including variable, function, class/instance, module/package.
- Names defined inside a function are local. Names defined outside all functions are global for that module, and are accessible by all functions inside the module (i.e., module-global scope).

- A name in Python is just a way to access a variable like in any other languages. However, Python is more flexible when it comes to the variable declaration.
- Declare a variable by just assigning a name to it.

```
 >>> a=2   # declare variable a
>>> id(a)   # returns memory address of variable a (integer value)
    140704266720960
>>> id(2) # returns the address of value
    140704266720960
>>> a=a+1  # increment the value of a  is now 3
>>> id(a)  #Address of variable a now stores value is 3
    140704266720992
>>> id(3)
    140704266720992
```



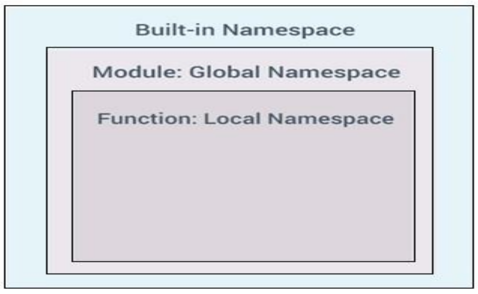
Namespaces:

What are namespaces in Python?

- A namespace is a collection of names (i.e., a space of names).A namespace is a simple system to control the names in a program.
- It ensures that names are unique and won't lead to any conflict.
- Python implements namespaces in the form of dictionaries. It maintains a name-to-object mapping where names act as keys and the objects as values.
- Multiple namespaces may have the same name but pointing to a different variable.
- Name (an unique identifier) + Space(related to scope).
- An example, a directory-file system structure in computers
- Real-time example, the role of a namespace is like a surname
- A namespace is a system to have a unique name for each and every object in Python.
- An object might be a variable or a method. a namespace (sometimes also called a context) is a naming system for making names unique to avoid ambiguity.
- A name might be of any Python method or variable and space depends upon the location from where is trying to access a variable or a method.

- Each module creates its own global namespace. These different namespaces are isolated. Hence, the same names that may exist in different modules do not collide.

Following diagram may help to clarify this concept.



A diagram of different namespaces in Python

As shown in the following figure, same object name can be present in multiple namespaces as isolation between the same name is maintained by their namespace

Some namespaces in Python: or Types of namespaces:

- Built In Name Space
- Global Name Space
- Enclosing Name Space
- Local Name Space

➢ Built-in names: this namespace contains built-in functions (e.g. abs(), cmp(), ...) and built-in exception names

➢ Global names of a module

➢ Local names in a function or method invocation

Built-in Namespace

- The built-in namespace contains the names of all of Python’s built-in objects. These are available at all times when Python is running.
- List the objects in the built-in namespace with the following command:
- >>> dir(__builtins__)
  ['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError',
- This namespace covers the built-in functions and built-in exception names.
- Python creates it as the interpreter starts and keeps it until you exit.
- Built-in names: this namespace contains built-in functions (e.g. abs(), cmp(), ...) and built-in exception names

Global Namespace

- The **global namespace** contains any names defined at the level of the main program.

- Python creates the global namespace when the main program body starts, and it remains in existence until the interpreter terminates.
- Global names of a module
- This namespace covers the names from various imported modules used in a project.
- Python creates this namespace for every module included in your program. It'll last until the program ends.

Local Namespace

- This namespace covers the local names inside a function.
- Python creates this namespace for every function called in a program.
- It remains active until the function returns.
- Local names within a function or method invocation
- The interpreter creates a new namespace whenever a function executes. That namespace is local to the function and remains in existence until the function terminates.

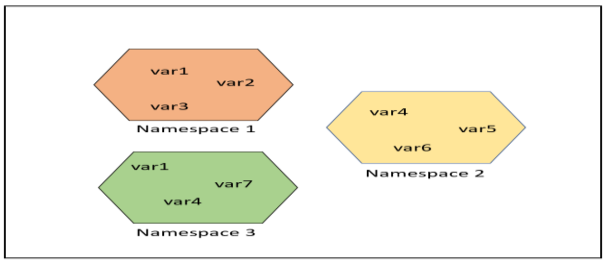
Lifetime of a namespace:

- A lifetime of a namespace depends upon the scope of objects, if the scope of an object ends, the lifetime of that namespace comes to an end.
- Hence, it is not possible to access inner namespace's objects from an outer namespace.

Example:

```
# var1 is in the global namespace
var1 = 5
def some_func():
     # var2 is in the local namespace
    var2 = 6
    def some_inner_func():
        # var3 is in the nested local
      # namespace
      var3 = 7
```

As shown in the following figure, same object name can be present in multiple namespaces as isolation between the same name is maintained by their namespace.



But in some cases, one might be interested in updating or processing global variable only, as shown in the following example, one should mark it explicitly as global and the update or process.

```
# Python program processing
# global variable
  count = 5
def some_method():
    global count
    count = count + 1
    print(count)
some_method()
```

Output:

```
6
```

Scope of Objects in Python:

- Scope refers to the coding region from which particular Python object is accessible.
- Hence one cannot access any particular object from anywhere from the code, the accessing has to be allowed by the scope of the object.

Lets take an example to have details understanding of the same:

```
 # Python program showing
# a scope of object
  def some_func():
    print("Inside some_func")
    def some_inner_func():
        var = 10
        print("Inside inner function, value of var:",var)
    some_inner_func()
    print("Try printing var from outer function: ",var)
some_func()
```

Output:

```
Inside some_func
Inside inner function, value of var: 10
Traceback (most recent call last):
  File "/home/1eb47bb3eac2fa36d6bfe5d349dfcb84.py", line 8, in
    some_func()
  File "/home/1eb47bb3eac2fa36d6bfe5d349dfcb84.py", line 7, in some_func
    print("Try printing var from outer function: ",var)
NameError: name 'var' is not defined
```

Builtin Modules- os, random, math, cmath, pprint, json, request, date, RegEx.

- The Python interpreter has a number of built-in functions. They are loaded automatically as the interpreter starts and are always available.
- a large number of pre-defined functions are also available as a part of libraries bundled with Python distributions.
- These functions are defined in modules. A module is a file containing definitions of functions, classes, variables, constants or any other Python objects.
- Contents of this file can be made available to any other program.
- Built-in modules are written in C and integrated with the Python interpreter.
- Each built-in module contains resources for certain system-specific functionalities such as OS management, disk IO, etc.
- The standard library also contains many Python scripts (with the .py extension) containing useful utilities.
- To display a list of all available modules, use the following command in the Python console:
- `>>> help('modules')`

  Please wait a moment while I gather a list of all available modules...

  ### Operating System(OS) Module in Python with Examples

  The OS module in python provides functions for interacting with the Operating System. OS, comes under Python’s standard utility modules.

  This module provides a portable way of using operating system dependent functionality.

  It is possible to automatically perform many operating system tasks.

  The OS module in Python provides functions for creating and removing a directory (folder),

  fetching its contents, changing and identifying the current directory, etc.

  Following are some functions in OS module:

1. os.name:

This function gives the name of the operating system dependent module imported. The following names have currently been registered: 'posix', 'nt', ‘os2’, ‘ce’, ‘java’ and ‘riscos’

```
>>> import os
>>> print(os.name)
```

Output:

```
nt
```

2.Creating Directory To make a new directory:

We can create a new directory using the mkdir() function from the OS module.

```
>>> import os
>>> os.mkdir("e:\\tempdir")
```

A new directory corresponding to the path in the string argument of the function will be created.

If we open D drive in Windows Explorer, we should notice tempdir folder created.

To change the name of, or rename, a directory:

3.Changing the Current Working Directory

first change the current working directory to a newly created one before doing any operations in it. This is done using the chdir() function.

```
>>> import os
>>> os.chdir('e:\\tempdir')
```

There is a getcwd() function in the OS module using which we can confirm if the current working directory has been changed or not.

```
>>> os.getcwd()
```

output:'e:\\tempdir'

Directory paths can also be relative.

If the current directory is set to D drive and then to tempdir without mentioning the preceding path, then also the current working directory will be changed to d:\tempdir.

```
>>> os.chdir("e:\\")
>>> os.getcwd()
'e:\\'
>>> os.chdir("tempdir")
>>> os.getcwd()
'e:\\tempdir'
>>>
```

In order to set the current directory to the parent directory use ".." as the argument in the chdir() function.

```
>>> os.chdir("tempdir")
>>> os.getcwd()
'e:\\tempdir'
>>> os.chdir("e:\\tempdir")
>>> os.getcwd()
'e:\\tempdir'
>>> os.chdir("..")
>>> os.getcwd()
'e:\\'
```

Removing a Directory

The rmdir() function in the OS module removes the specified directory either with an absolute or relative path.

However, we can not remove the current working directory.

Also, for a directory to be removed, it should be empty.

For example, tempdir will not be removed if it is the current directory.

We have to change the current working directory and then remove tempdir.

```
>>> os.chdir("tempdir")
>>> os.getcwd()
'e:\\tempdir'
os.rmdir("e:\\tempdir")
PermissionError: [WinError 32] The process cannot access the file because it is being used by  another process: 'e:\\tempdir'
>>> os.chdir("..")
```

List Files and Sub-directories

The listdir() function returns the list of all files and directories in the specified directory.

```
>>> os.listdir("c:\python38")
['2.py', '21.py', '22.py', '2cc.py', '2fr.py', '3.py', 'aa.py', 'DLLs', 'Doc', 'el.py'’]
```

If we don't specify any directory, then list of files and directories in the current working directory will be returned.

```
>>> os.listdir("e:\pp")
['block.py', 'cse.py', 'emp.py', 'first.py', 'madhu.py', 'madhu1.py', 'nams.py', 'sampl.py', 'sample.py', 'sec.py', 'smp.py']
>>> os.listdir()
```

```
['$RECYCLE.BIN', '2017_2021 BATCH DATA', '2019_20_personal aprisal', '2020_2021 PYTHON COURSE FILE', 'AI DEEP', 'AI FDP 11 -15 MAY',]
```

Python - Random Module

Functions in the random module depend on a pseudo-random number generator function random(),which generates a random float number between 0.0 and 1.0.

random.random():

Generates a random float number between 0.0 to 1.0. The function doesn't need any arguments.

```
>>>import random
>>>random.random()   # float in [0,1)
0.645173684807533
```

random.randint():

Returns a random integer between the specified integers.

```
>>>import random
>>>random.randint(1,100)  # int in [1,100]
95
>>>random.randint(1,100)   # int in [1,100]
49
```

random.randrange():

Returns a randomly selected element from the range created by the start,stop and step arguments. The value of start is 0 by default. Similarly, the value of step is 1 by default.

```
>>>random.randrange(1,10)  # From range(10), i.e., 1 to 9
2
>>>random.randrange(1,10,2)
5
>>>random.randrange(0,101,10)
80
```

random.choice():

Returns a randomly selected element from a non-empty sequence.An empty sequence as argument raises an IndexError.

```
>>>import random
>>>random.choice('computer')          # Pick one value from the given list
't'
>>>random.choice([12,23,45,67,65,43])
45
>>>random.choice((12,23,45,67,65,43))
```

67

random.shuffle():This functions randomly reorders the elements in a list.

```
>>>numbers=[12,23,45,67,65,43]
>>>random.shuffle(numbers)   #shuffle the values in list
>>>numbers
[23, 12, 43, 65, 67, 45]
>>>random.shuffle(numbers)
>>>numbers
[23, 43, 65, 45, 12, 67]
```

Python math Module: What is math module in Python?

- Python has a built-in module that you can use for mathematical tasks. The math module is a standard module in Python and is always available.
- In python a number of mathematical operations can be performed with ease by importing a module named 'math' which defines various functions which makes our tasks easier.
- Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc.
- In addition, two mathematical constants are also defined in Math module.
- For example: # Square root calculation

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

This module does not support complex data types. The math module has a set of methods or functions and constants.

Functions in Python Math Module:List of Functions in Python Math Module

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
fabs(x)	Returns the absolute value of x
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
fmod(x, y)	Returns the remainder when x is divided by y
frexp(x)	Returns the mantissa and exponent of x as the pair (m, e)
fsum(iterable)	Returns an accurate floating point sum of values in the iterable
isfinite(x)	Returns True if x is neither an infinity nor a NaN
isinf(x)	Returns True if x is a positive or negative infinity
modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns e**x
log(x[, b])	Returns the logarithm of x to the base b (defaults to e)
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
cos(x)	Returns the cosine of x
tanh(x)	Returns the hyperbolic tangent of x

Math methods or functions**:**

math.ceil()   Rounds a number up to the nearest integer

```
# Import math library
```

```
>>>import  math
# Round a number upward to its nearest integer
>>> print(math.ceil(20.67))
21
>>> print(math.ceil(-5.3))
-5
>>> print(math.ceil(10.0))
10
```

Python math.floor() Method : Round numbers down to the nearest integer:

```
# Import math library
>>>import math
# Round numbers down to the nearest integer
>>> import math
>>> print(math.floor(0.6))
0
>>> print(math.floor(5.3))
5
>>> print(math.floor(-5.3))
-6
```

Python math.dist() Method

Example :Find the Euclidean distance between one and two dimensional points:

```
# Import math Library
>>>import math
p = [3]
q = [1]
# Calculate Euclidean distance
print (math.dist(p, q))
output: 2.0
>>>
 example 2:
# Import math Library
import math
p = [3, 3]
q = [6, 12]
# Calculate Euclidean distance
print (math.dist(p, q))
output: 9.486832980505138
```

Python math.exp() Method :Return 'E' raised to the power of different numbers:

```
#Import math Library
import math
#find the exponential of the specified value
print(math.exp(65))
print(math.exp(-6.89))
```

output:

```
1.6948892444103338e+28
0.0010179138409954387
```

Python math.factorial() Method : Find the factorial of a number:

```
#Import math Library
import math
#Return factorial of a number
>>>print(math.factorial(9))
362880
>>>print(math.factorial(6))
720
```

Python math.isclose() Method : Check whether two values are close to each other, or not:

```
#Import math Library
import math
#compare the closeness of two values
print(math.isclose(1.233, 1.4566))
print(math.isclose(1.233, 1.233))
print(math.isclose(1.233, 1.24))
print(math.isclose(1.233, 1.233000001))
```

output:

```
False
True
False
True
```

Python math.log() Method: Find the natural logarithm of different numbers.

The math.log() method returns the natural logarithm of a number, or the logarithm of number to base.

Syntax

math.log(x, base)

Parameter Values

Parameter	Description
x	Required. Specifies the value to calculate the logarithm for. If the value is 0 or a negative number, it returns a ValueError. If the value is not a number, it returns a TypeError

```
# Import math Library
import    math
# Return the natural logarithm of different numbers
print(math.log(2.7183))
print(math.log(2))
print(math.log(1))
```

output:

```
1.0000066849139877
0.6931471805599453
0.0
```

Python math.log10() Method : Find the base-10 logarithm of different numbers

```
# Import math Library
import math
# Return the base-10 logarithm of different numbers
print(math.log10(2.7183))
print(math.log10(2))
print(math.log10(1))
```

output:

```
0.43429738512450866
0.3010299956639812
0.0
```

Python math.log2() Method

Find the base-2 logarithm of different numbers

```
# Import math Library
import math
# Return the base-2 logarithm of different numbers
print(math.log2(2.7183))
print(math.log2(2))
print(math.log2(1))
```

output:

```
1.4427046851812222
1.0
```
0.0

Math Constants

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)

Python math.e Constant :Print the Euler's number:

```
# Import math Library
import math
# Print the value of Euler e
print (math.e)
```
output:
```
2.718281828459045
```
Definition and Usage

The math.e constant returns the Eular's number: 2.718281828459045.

Syntax

math.e

Technical Details

Return Value: A float value, 2.718281828459045, representing the mathematical constant e

Python math.pi Constant : Print the value of PI:

```
# Import math Library
import math
# Print the value of pi
print (math.pi)
```
output:
```
3.141592653589793
```
Definition and Usage

The math.pi constant returns the value of PI: 3.141592653589793.

Note: Mathematically PI is represented by π.

Syntax

math.pi

cmath — Mathematical functions for complex numbers

- Python can also handle complex numbers and its associated functions using the file 'cmath'.
- This module is always available. It provides access to mathematical functions for complex numbers.
- The functions in this module accept integers, floating-point numbers or complex numbers as arguments.

- They will also accept any Python object that has either a __complex__() or a __float__() method:
- these methods are used to convert the object to a complex or floating-point number,respectively, and the function is then applied to the result of the conversion.

Converting real numbers to complex number

An complex number is represented by ' x + yj '. Python converts the real numbers x and y into complex using the function complex(x,y).

The real part can be accessed using the function real() and imaginary part can be represented by imag().

- Conversions to and from polar coordinates
- Power and logarithmic functions
- Trigonometric functions
- Hyperbolic functions
- Classification functions
- Constants

Conversions to and from polar coordinates:

A Python complex number z is stored internally using rectangular or Cartesian coordinates. It is completely determined by its real part z.real and its imaginary part z.imag.
In other words:

z == z.real + z.imag*1j

Polar coordinates give an alternative way to represent a complex number.

Python cmath.polar() Method :Convert a complex number to polar coordinates form:

```
#import cmath for complex number operations
import cmath
#find the polar coordinates of complex number
print (cmath.polar(2 + 3j))
print (cmath.polar(1 + 5j))
```

output:

```
(3.605551275463989, 0.982793723247329)
(5.0990195135927845, 1.373400766945016)
```

Definition and Usage...

The cmath.polar() method converts a complex number to polar coordinates. It returns a tuple of modulus and phase.In polar coordinates, a complex number is defined by modulus r and phase angle phi.

Syntax : cmath.polar(x)

Parameter Values

Parameter	Description
x	Required. A number to find polar coordinates of

Python cmath.phase() Method : Find the phase of a complex number:

```
#Import cmath Library
import cmath
#print phase of some given parameters
print (cmath.phase(2 + 3j))
```

output:

```
0.982793723247329
```

Definition and Usage :

The cmath.phase() method returns the phase of a complex number.A complx number can be expressed in terms of its magnitude and angle.This angle is between vector (representing complex number) and positive x-axis is called Phase.

Note: Output is always between -π and π.

Syntax
cmath.phase(x)
Parameter Values

Parameter	Description
x	Required. The number to find the phase of

Conversions to and from polar coordinates
----

cmath.phase(x)
Return the phase of x (also known as the argument of x), as a float.phase(x) is equivalent to math.atan2(x.imag, x.real).The result lies in the range [-π, π], and the branch cut for this operation lies along the negative real axis, continuous from above.
cmath.polar(x)
Return the representation of x in polar coordinates.
Returns a pair (r, phi) where r is the modulus of x and phi is the phase of x.
polar(x) is equivalent to (abs(x), phase(x)).
cmath.rect(r, phi)
Return the complex number x with polar coordinates r and phi.
Equivalent to r * (math.cos(phi) + math.sin(phi)*1j).

Power and logarithmic functions
----

cmath.exp(x) : Return the exponential value e**x.
cmath.log(x[, base]) : Returns the logarithm of x to the given base.
If the base is not specified, returns the natural logarithm of x.
There is one branch cut, from 0 along the negative real axis to -∞, continuous from above.
Changed in version 2.4: base argument added.
cmath.log10(x) :Return the base-10 logarithm of x. This has the same branch cut as log().
cmath.sqrt(x) :Return the square root of x. This has the same branch cut as log().

Trigonometric functions
----

cmath.acos(x) :Return the arc cosine of x. There are two branch cuts: One extends right from 1 along the real axis to ∞, continuous from below. The other extends left from -1 along the real axis to -∞, continuous from above.
cmath.asin(x) :Return the arc sine of x. This has the same branch cuts as acos().
cmath.atan(x) :Return the arc tangent of x. There are two branch cuts: One extends from 1j along the imaginary axis to ∞j, continuous from the right. The other extends from -1j along the imaginary axis to -∞j, continuous from the left.
Changed in version 2.6: direction of continuity of upper cut reversed
cmath.cos(x) :Return the cosine of x.
cmath.sin(x) :Return the sine of x.
cmath.tan(x) :Return the tangent of x.

Hyperbolic functions
cmath.acosh(x) :Return the inverse hyperbolic cosine of x. There is one branch cut, extending left from 1 along the real axis to -∞, continuous from above.
cmath.asinh(x) :Return the inverse hyperbolic sine of x. There are two branch cuts: One extends from 1j along the imaginary axis to ∞j, continuous from the right. The other extends from -1j along the imaginary axis to -∞j, continuous from the left.
cmath.atanh(x) :Return the inverse hyperbolic tangent of x. There are two branch cuts: One extends from 1 along the real axis to ∞, continuous from below. The other extends from -1 along the real axis to -∞, continuous from above.
cmath.cosh(x) :Return the hyperbolic cosine of x.
cmath.sinh(x):Return the hyperbolic sine of x.
cmath.tanh(x):Return the hyperbolic tangent of x.
Classification functions

cmath.isinf(x) :Return True if the real or the imaginary part of x is positive or negative infinity.
cmath.isnan(x) :Return True if the real or imaginary part of x is not a number (NaN).

Constants

cmath.pi :The mathematical constant π, as a float.
cmath.e :The mathematical constant e, as a float.

pprint — Data pretty printer

The pprint module provides a capability to "pretty-print" arbitrary Python data structures in a well-formatted and more readable way!

```
s2={1,2,3,4,5}
type(s2)
dir(s2)

for eo in dir(s2):
        print(eo)

from pprint import pprint

pprint(dir(s2))
```

****

API KEY:&apikey=544bb8f2

****

```
# A python code without pprint
import requests
#from pprint import pprint
def omdb(title):
   url = "http://www.omdbapi.com/?t=hulk&apikey=6ffb04ae"
   resp = requests.get(url, params = {'title': title})
   return resp.json()

# calling geocode function
data =omdb('hulk')
# pretty-printing json response
print(data)


# A python code with pprint
import requests
from pprint import pprint
def omdb(title):
   url = "http://www.omdbapi.com/?t=hulk&apikey=6ffb04ae"
   resp = requests.get(url, params = {'title': title})
   return resp.json()

# calling geocode function
data =omdb('hulk')
# pretty-printing json response
pprint(data)
```

JSON (JavaScript object notation) format

- Python has a built-in package called json, which can be used to work with JSON data.

- JSON stands for JavaScript Object Notation. JSON is a popular data format used for representing structured data .It is a lightweight data interchange format.
- JSON is syntax for storing and exchanging data. JSON is text, written with JavaScript object notation.
- This format is used for data exchange between the web server and clients. or It's common to transmit and receive data between a server and web application in JSON format.

- The json module provides dumps() and loads() function for serialization of Python

- dumps() − This function converts the object into JSON format.
- loads() − This function converts a JSON string back to Python object.

In Python, JSON exists as a string. For example:

```
person = ' {"name": "Ravi",
                  "languages": ["Python", "Java"]}'

emp='{ "ename":"smith",}'
```

Import json Module

- To work with JSON (string, or file containing JSON object), you can use Python's json module.You need to import the module before you can use it.
  import json

Parse JSON in Python

- The json module makes it easy to parse JSON strings and files containing JSON object.
- Example: Import the json module:
  import json
- The json module defines load() and dump() functions to write JSON data to a file like object which may be a disk file or a byte stream and read data back from them.

1. Parse JSON - Convert from JSON to Python

- If you have a JSON string, you can parse it by using the json.loads() method. The result will be a Python dictionary.
- load():This function loads JSON data from the file and returns Python object from it.

Example : Convert from JSON to Python:

```
import  json
# #json string format to python dictionary format conversion
emp='{"ename":"madhu","design":"associate professor","dept":"cse","esal":"20000"}'
# parse x:
em_lst=json.loads(emp)
# the result is a Python dictionary:
print('The Python format is Dictionary is...\n',em_lst)
```

output: Test Case:1

```
The Python format is Dictionary is...
 {'ename': 'madhu', 'design': 'associate professor', 'dept': 'cse', 'esal': '20000'}
```

Example-2

```
# #json string format to python dictionary format conversion
import json
emp='{"ename":"madhu","design":"associate professor","dept":"cse","esal":"20000"}'
em_lst=json.loads(emp)
print('The Python format is Dictionary is...\n',em_lst['design'])
```

Output: Test Case:2

```
The Python format is Dictionary is...
 associate professor
```

2. Convert from Python to JSON

- If you have a Python object(dictionary) convert it into a JSON string by using the json.dumps() method.
- dump():This function writes JSONed Python object data to a file

Example1: Convert from Python(dictionary) to JSON format(string):

```
import  json

#  Python object (dictionary) to json string format
stu={'sname':'smith','branch':'cse','year':'II CSE ','City':'vzm'}
# convert into JSON:
stul=json.dumps(stu)
# the result is a JSON string:
print('The JSON  string format is...\n',stul)
```

output:

```
The JSON  string format is...
 {"sname": "smith", "branch": "cse", "year": "II CSE ", "City": "vzm"}
```

REQUEST: Python Requests Module

- Make a request to a web page, and print the response text:
- Requests library is one of the integral part of Python for making HTTP requests to a specified URL.Whether it be REST APIs or Web Scrapping, requests is must to be learned for proceeding further with these technologies.
- When one makes a request to a URI, it returns a response.
- Python requests provide inbuilt functionalities for managing both the request and response.

Definition and Usage

The requests module allows you to send HTTP requests using Python.The HTTP request returns a Response Object with all the response data (content, encoding, status, etc).

Syntax : requests.methodname(params)

Method	Description
delete(url, args)	Sends a DELETE request to the specified url
get(url, params, args)	Sends a GET request to the specified url
head(url, args)	Sends a HEAD request to the specified url
patch(url, data, args)	Sends a PATCH request to the specified url

post(url, data, json, args)	Sends a POST request to the specified url
put(url, data, args)	Sends a PUT request to the specified url
request(method, url, args)	Sends a request of the specified method to the specified url

Why learn Python requests module?

To play with web, Python Requests is must.Whether it be hitting APIs, downloading entire facebook pages, and much more cool stuff, one will have to make a request to the URL.

Requests play a major role is dealing with REST APIs, and Web Scrapping.
Example

```
import requests

# Making a GET request
r = requests.get('https://www.flipkart.com/')

# check status code for response received
r
# success code - 200
print(r)

# print content of request
print(r.content)
```
****
json

```
E:\pp>python
>>> import requests as r
>>> import json as j
>>> mv=r.get("http://www.omdbapi.com/?t=hulk&apikey=6ffb04ae")
>>> mv
<Response [200]>
>>> mv.text
>>> inf=j.loads(mv.text)
>>> inf
>>> type(inf)
<class 'dict'>
>>> inf['Tile']
>>> inf['Title']
'Hulk'
```
DATE & TIME: **Python Datetime**

Python Dates
A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Example : Import the datetime module and display the current date:

```
>>> import datetime
>>> x = datetime.datetime.now()
>>> x
datetime.datetime(2021, 1, 7, 21, 30, 15, 885098)
>>>
```

The date contains year, month, day, hour, minute, second, and microsecond.The datetime module has many methods to return information about the date object.

Example:Return the year and name of weekday:

```
import datetime

>>> x = datetime.datetime.now()
>>> print(x.year)
2021
>>> print(x.strftime("%A"))
Thursday
```

Creating Date Objects

To create a date, we can use the datetime() class (constructor) of the datetime module.The datetime() class requires three parameters to create a date: year, month, day.
Example: Create a date object:

```
>>> x = datetime.datetime(2021, 1, 17)
>>> print(x)
2021-01-17 00:00:00
```

The datetime() class also takes parameters for time and timezone (hour, minute, second, microsecond, tzone), but they are optional, and has a default value of 0, (None for timezone).
The strftime() Method:The datetime object has a method for formatting date objects into readable strings.The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:
Example: Display the name of the month:

```
import datetime
>>> x = datetime.datetime(2021, 1, 1)
>>> print(x.strftime("%B"))
January
>>>
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed

```
import datetime
>>> x = datetime.datetime(2021, 1, 1)
>>> print(x.strftime("%B"))
January
>>> x = datetime.datetime.now()
>>> print(x.strftime("%a"))
Thu
>>> print(x.strftime("%A"))
Thursday
>>>
```

%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18

%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100
%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%

Python RegEx(Regular Expression)

- The task of searching and extracting is so common that python has a very powerful library called regular expressions
- A Regular Expression (RegEx) is a sequence of characters that defines a search pattern.
- RegEx can be used to check if a string contains the specified search pattern.
- A regular expression in a programming language is a special text string used for describing a search pattern.
- It is extremely useful for extracting information from text such as code, files, log, spreadsheets or even documents.
- While using the regular expression the first thing is to recognize is that everything is essentially a character, and we are writing patterns to match a specific sequence of characters also referred as string
- string includes digits and punctuation and all special characters like $#@!%, etc.

Example
string: a b a b a b a b a a b b a a a b b b a b b a c c c c a d v c
pattern: a a a b or ba or a

string: 'coding in python is very easy happy coding in python'
pattern: py
[^coding-python]$
ex:^a...s$

The above code defines a RegEx pattern. The pattern is: any five letter string starting with **a** and ending with **s**.

RegEx Module :Python has a built-in package called re, which can be used to work with Regular Expressions. Import the re module:

Import Regular Expression Syntax : **import re**

"re" module included with Python primarily used for string searching and manipulation
The re module provides support for regular expression (or regex in short).

RegEx Functions :The re module offers a set of functions that allows us to search a string for a match:

Function	Description
findall()	Returns a list containing all matches
search()	Returns a Match object if there is a match anywhere in the string
split()	Returns a list where the string has been split at each match
sub()	Replaces one or many matches with a string

1.The findall() Function or re.findall()

The findall() or re.findall() function returns a list containing all matches.
The list contains the matches in the order they are found.
If no matches are found, re.findall() returns an empty list.

Metacharacters

Metacharacters are characters with a special meaning:

Character	Description	Example
[]	A set of characters	"[a-k]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character) A...l	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences [0-more]	"aix*"
+	One or more occurrences [1-more]	"aix+"

a*=0,1,2,..n
a+=1,2,..n

{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Example 1:[] A set of characters

```
import re
txt = "The rain in Spain"
#Find all lower case characters alphabetically between "a" and "m":
x = re.findall("[a-m]", txt)
```

```
print(x)
```
output: ['h', 'e', 'a', 'i', 'i', 'a', 'i']

Example 2 :\d :Find all digit characters

```
import re
txt = "That 90 will be 59 dollars"
#Find all digit characters:
x = re.findall("\d", txt)
print(x)
```
output : ['9', '0', '5', '9']

Example 3: . any character except newline

```
import re
txt = "hello '\n' world"
#Search for a sequence that starts with "he", followed by two (any) characters, and an "o":
x = re.findall("wo..d", txt)print(x)
```
output: ['world']

Example 4: ^ start with

```
import re
txt = "Python programming very useful "
#Check if the string starts with 'Python':
x = re.findall("^Python", txt)
if x:
  print("Yes, the string starts with 'Python'")
else:
  print("No match")
```
output:Yes, the string starts with 'Python'

Example 5: $ endwith

```
import re
txt = "happy coding in python is easy"
#Check if the string ends with 'world':
x = re.findall("easy$", txt)
if x:
  print(" Yes, the string ends with 'easy' ")
else:
  print("No match")
```
output: Yes, the string ends with 'easy'

Example 6: * Zero or more occurrences

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "ai" followed by 0 or more "x" characters:
x = re.findall("aix*", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```

output: ['ai', 'ai', 'ai', 'ai'] Yes, there is at least one match!

Example 7: + One or more occurrence

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "ai" followed by 1 or more "x" characters:
x = re.findall("aix+", txt)
print(x)
```

```
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```
output: [] No match:

Example 8: {} Exactly the specified number of occurrences

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains "a" followed by exactly two "l" characters:
x = re.findall("al{2}", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```
output: ['all'] Yes, there is at least one match!

Example 9: | Either or

```
import re
txt = "The rain in Spain falls mainly in the plain!"
#Check if the string contains either "falls" or "stays":
x = re.findall("falls|stays", txt)
print(x)
if x:
  print("Yes, there is at least one match!")
else:
  print("No match")
```
output: ['falls'] Yes, there is at least one match!

2.The search() Function

The search() function searches the string for a match, and returns a Match object if there is a match.If there is more than one match, only the first occurrence of the match will be returned:

Example

Search the string to see if it starts with "The" and ends with "Spain":

```
import re
#Check if the string starts with "The" and ends with "Spain":
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
if x:
  print("YES! We have a match!")
else:
  print("No match")
```
output: YES! We have a match!

Example..

```
import re
pattern = '^P...n$'
test_string = 'Pyhon'
result = re.match(pattern, test_string)
if result:
  print("Search Successful...")
else:
  print("Search Unsuccessful.....:")
```
output: Search Successful...

Example

Search for the first white-space character in the string:

```
import re
txt = "The rain in Spain"
x = re.search("\s", txt)
print("The first white-space character is located in position:", x.start())
```

output:The first white-space character is located in position: 3

Example: Make a search that returns no match:

```
import re
txt = "happy coding in programming"
x = re.search("python", txt)
print(x)
```

output: None

3.The split() Function

The split() function returns a list where the string has been split at each match:

Example: Split at each white-space character:

```
import re
#Split the string at every white-space character:
txt = "welcome to python coding"
x = re.split("\s", txt)
print(x)
```

output: ['welcome', 'to', 'python', 'coding']

You can control the number of occurrences by specifying the maxsplit parameter:

Example : Split the string only at the first occurrence:

```
import re
txt = 'welcome to python coding'
x = re.split("\s", txt, 1)
print(x)
```

output: ['welcome', 'to python coding']

4.The sub() Function :The sub() function replaces the matches with the text of your choice:

Example: Replace every white-space character with the number *@:

```
import re
#Replace all white-space characters with the digit "*@":
txt = "happy coding in python programming"
x = re.sub("\s", "*@", txt)
print(x)
```

output:happy*@coding*@in*@python*@programming

Example:

```
import re
#Replace all white-space characters with the digit "*@":
txt = "jack happy coding in python and jill  in yellow lab"
x = re.sub("jack", "jackjill", txt)
#y = re.sub("jill", "jilljack", txt)
print(x)
#print(y)
```

output:

```
jackjill happy coding in python and jill  in yellow lab
jack happy coding in python and jilljack  in yellow lab
```

You can control the number of replacements by specifying the count parameter:

Example

Replace the first 2 occurrences:

```
import re
txt = 'coding in python easy'
x = re.sub("\s", "*", txt, 2)
```

print(x)
output:
coding*in*python easy

Match Object : A Match Object is an object containing information about the search and the result.
Note: If there is no match, the value None will be returned, instead of the Match Object.
Example
Do a search that will return a Match Object:

```
import re
#The search() function returns a Match object:
txt = 'happy coding in python'
x = re.search("python", txt)
print(x)
```

output:

```
-----------
<re.Match object; span=(16, 22), match='python'>
```

The Match object has properties and methods used to retrieve information about the search, and the result:

.span() returns a tuple containing the start-, and end positions of the match.
.string returns the string passed into the function
.group() returns the part of the string where there was a match

Example
Print the position (start- and end-position) of the first match occurrence.
The regular expression looks for any words that starts with an upper case "S":

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.span())
```

output:

```
(12, 17)
```

When r or R prefix is used before a regular expression, it means raw string.
Example
Print the string passed into the function:

```
import re
txt = "The rain in Spain"
x = re.search(r"\bS\w+", txt)
print(x.string)
```

output:

```
The rain in Spain
```

Example
Print the part of the string where there was a match.The regular expression looks for any words that starts with an upper case "P":

```
import re
txt = "welcome to python Programming "
x = re.search(r"\bP\w+", txt)
print(x.group())
```

output:

```
---------
Programming
```

Note: If there is no match, the value None will be returned, instead of the Match Object.