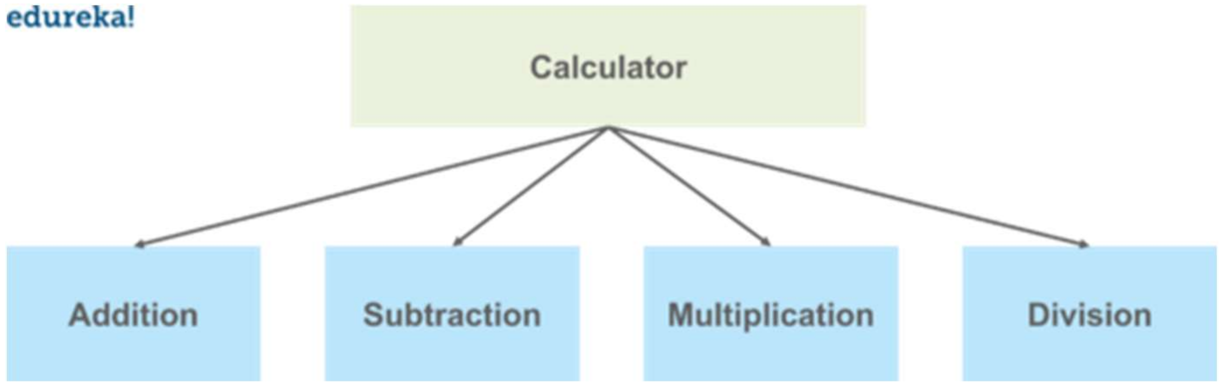
Unit 5: Modules

What are modules?

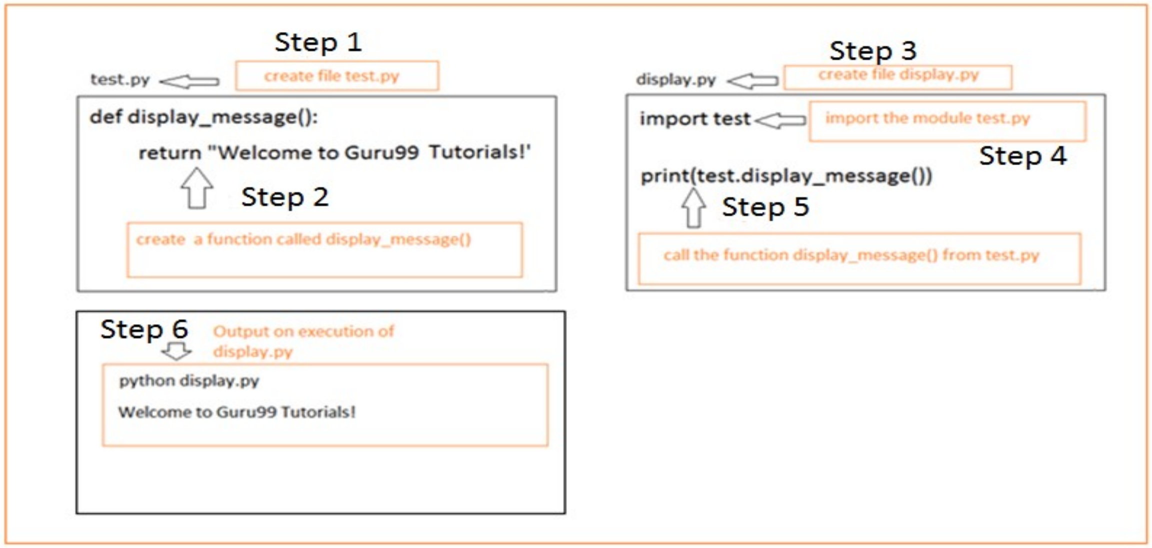
- A Python module is a file containing Python definitions and statements.
- A module can define functions, classes, and variables.
- A module can also include runnable code.
- Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.

- In Python, **Modules** are simply files with the ".py" extension containing Python code that can be imported inside another Python Program.
- In simple terms, we can consider a module to be the same as a code library or a file that contains a set of functions that you want to include in your application.
- With the help of modules, we can organize related functions, classes, or any code block in the same file.
- So, It is considered a best practice while writing bigger codes for production-level projects in Data Science is to split the large Python code blocks into modules containing up to 300–400 lines of code.
- The module contains the following components:
  - **Definitions and implementation of classes,**
  - **Variables, and**
  - **Functions that can be used inside another program.**


Calculator
Addition
Subtraction
Multiplication
Division

How to create Python Modules?

- To create a module, we have to save the code that we wish in a file with the file extension **".py"**. Then, the name of the Python file becomes the name of the module.


Step 1
test.py
create file test.py
Step 2
create a function called display_message()
Step 3
display.py
create file display.py
import the module test.py
Step 4
Step 5
call the function display_message() from test.py
Step 6
Output on execution of display.py

```
def display_message():
      return "Welcome to Guru99 Tutorials!'
```

```
import test

print(test.display_message())
```

```
python display.py
Welcome to Guru99 Tutorials!
```

Importing module

- Now we can use the module we just created, by using the import statement
- We use the import keyword to do this. To import our previously defined module we type the following in the Python prompt.
- **Syntax:-**
- **Import module name**
- This does not import the names of the functions defined in module directly in the current symbol table.
- It only imports the module name there.
- Using the module name we can access the function using the dot . operator.
- **For example: module name. Function name()**

Python import statement

- Module can be imported in following three ways
- 1. We can import a module using the `import` statement and access the definitions inside it using the dot operator as described above. Here is an example.

```
# to import standard module math
import math
print("The value of pi is", math.pi)
```

- **2. Import with renaming**
- We can import a module by renaming it as follows:
- **# import module by renaming it**

```
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.

- **3. Python from...import statement**
- We can import specific names from a module without importing the module as a whole. Here is an example.
- **# 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**

- **Import all names**
- We can import all names(definitions) from a module using the following construct:
- **# 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.

Python Module Search Path

- While importing a module, Python looks at several places.
- Interpreter first looks for a built-in module.
- Then(if built-in module not found), Python looks into a list of directories defined in sys.path.
- The search is in this order.
  - The current directory
  - PYTHONPATH (an environment variable with a list of directories).
  - The installation-dependent default directory configured at the time Python is installed

Reloading a module

- The Python interpreter imports a module only once during a session. This makes things more efficient.
- Now if our module changed during the course of the program, we would have to reload it.
- One way to do this is to restart the interpreter. But this does not help much.
- Python provides a more efficient way of doing this.
- We can use the reload() function inside the imp module to reload a module.

Math Module

- Sometimes when working with some kind of financial or scientific projects it becomes necessary to implement mathematical calculations in the project.
- Python provides the **math module** to deal with such calculations. Math module provides functions to deal with both basic operations such as addition(+), subtraction(-), multiplication(*), division(/) and advance operations like trigonometric, logarithmic, exponential functions.

Constants provided by the math module

- Math module provides various the value of various constants like pi, tau. Having such constants saves the time of writing the value of each constant every time we want to use it and that too with great precision. Constants provided by the math module are –
- **Euler’s Number**
- **Pi**
- **Tau**
- **Infinity**
- **Not a Number (NaN)**

1.Euler’s Number

- The **math.e** constant returns the Euler’s number: 2.71828182846.
- **Syntax:** math.e

2. Pi

- You all must be familiar with pi. The pi is depicted as either 22/7 or 3.14. **math.pi** provides a more precise value for the pi.
- **Syntax:** math.pi

3. Tau

- **Tau** is defined as the ratio of the circumference to the radius of a circle. The **math.tau** constant returns the value tau: 6.283185307179586.
- **Syntax:** math.tau

4. Infinity

- Infinity basically means something which is never-ending or boundless from both directions i.e. negative and positive.
- It cannot be depicted by a number.
- The **math.inf** constant returns of positive infinity. For negative infinity, use **-math.inf**
- **Syntax:** math.inf

5. NaN

- The **math.nan** constant returns a floating-point nan (Not a Number) value. This value is not a legal number.
- The nan constant is equivalent to float(“nan”).

Numeric Functions

- **Finding the ceiling and the floor value**
- Ceil value means the smallest integral value greater than the number and the floor value means the greatest integral value smaller than the number. This can be easily calculated using the **ceil()** and **floor()** method respectively.

- # importing "math" for mathematical operations

```
import math
a = 2.3
# returning the ceil of 2.3
print ("The ceil of 2.3 is : ", end="")
print (math.ceil(a))
# returning the floor of 2.3
print ("The floor of 2.3 is : ", end="")
print (math.floor(a))
```

- **Finding the factorial of the number**
- Using the **factorial()** function we can find the factorial of a number in a single line of the code. An error message is displayed if number is not integral.
- # importing "math" for mathematical operations

```
import math
a = 5
# returning the factorial of 5
print("The factorial of 5 is : ", end="")
print(math.factorial(a))
```

- **Finding the GCD**
- **<u>gcd()</u>** function is used to find the greatest common divisor of two numbers passed as the arguments.

```
# importing "math" for mathematical operations
import math
a = 15
b = 5
# returning the gcd of 15 and 5
print ("The gcd of 5 and 15 is : ", end="")
print (math.gcd(b, a))
```

- **Finding the absolute value**
- **<u>fabs()</u>** function returns the absolute value of the number.

```
# importing "math" for mathematical operations
import math
a = -10
# returning the absolute value.
print ("The absolute value of -10 is : ", end="")
print (math.fabs(a))
```

Logarithmic and Power Functions

- Power functions can be expressed as x^n where n is the power of x whereas logarithmic functions are considered as the inverse of exponential functions.
- **Finding the power of exp**
- **exp() method** is used to calculate the power of e i.e.   or we can say exponential of y
- import math

```
# initializing the value
test_int = 4
test_neg_int = -3
test_float = 0.00
# checking exp() values
# with different numbers
print (math.exp(test_int))
print (math.exp(test_neg_int))
print (math.exp(test_float))
```

- **Finding the power of a number**
- **pow()** function computes x**y. This function first converts its arguments into float and then computes the power.
- # Python code to demonstrate pow()
- # version 1
- print ("The value of 3**4 is : ",end="")
- # Returns 81
- print (pow(3,4))

Finding the Logarithm

•**log()** function returns the logarithmic value of a with base b. If the base is not mentioned, the computed value is of the natural log.

•**log2(a)** function computes value of log a with base 2. This value is more accurate than the value of the function discussed above.

•**log10(a)** function computes value of log a with base 10. This value is more accurate than the value of the function discussed above.

```
# importing "math" for mathematical operations
import math
# returning the log of 2,3
print ("The value of log 2 with base 3 is : ", end="")
print (math.log(2,3))
# returning the log2 of 16
print ("The value of log2 of 16 is : ", end="")
print (math.log2(16))
# returning the log10 of 10000
print ("The value of log10 of 10000 is : ", end="")
print (math.log10(10000))
```

- **Finding the Square root**
- **sqrt()** function returns the square root of the number.
- # import the math module
- import math
- # print the square root of 0
- print(math.sqrt(0))
- # print the square root of 4
- print(math.sqrt(4))
- # print the square root of 3.5
- print(math.sqrt(3.5))

Trigonometric and Angular Functions

- You all must know about Trigonometric and how it may become difficult to find the values of sine and cosine values of any angle.
- Math module provides built-in functions to find such values and even to change the values between degrees and radians.
- **Finding sine, cosine, and tangent**
- **sin(), cos(), and tan()** functions returns the sine, cosine, and tangent of value passed as the argument. The value passed in this function should be in **radians**.

- # importing "math" for mathematical operations
- import math
- a = math.pi/6
- # returning the value of sine of pi/6
- print ("The value of sine of pi/6 is : ", end="")
- print (math.sin(a))
- # returning the value of cosine of pi/6
- print ("The value of cosine of pi/6 is : ", end="")
- print (math.cos(a))
- # returning the value of tangent of pi/6
- print ("The value of tangent of pi/6 is : ", end="")
- print (math.tan(a))

- **Converting values from degrees to radians and vice versa**
- **degrees()** function is used to convert argument value from radians to degrees.
- **radians()** function is used to convert argument value from degrees to radians.

```
# importing "math" for mathematical operations
import math
a = math.pi/6
b = 30
# returning the converted value from radians to degrees
print ("The converted value from radians to degrees is : ", end="")
print (math.degrees(a))
# returning the converted value from degrees to radians
print ("The converted value from degrees to radians is : ", end="")
print (math.radians(b))
```

Function Name	**Description**
ceil(x)	Returns the smallest integral value greater than the number
copysign(x, y)	Returns the number with the value of ‘x’ but with the sign of ‘y’
fabs(x)	Returns the absolute value of the number
factorial(x)	Returns the factorial of the number
floor(x)	Returns the greatest integral value smaller than the number
gcd(x, y)	Compute the greatest common divisor of 2 numbers

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 the precise floating-point value of sum of elements in an iterable
isfinite(x)	Check whether the value is neither infinity not Nan
isinf(x)	Check whether the value is infinity or not
isnan(x)	Returns true if the number is “nan” else returns false

modf(x)	Returns the fractional and integer parts of x
trunc(x)	Returns the truncated integer value of x
exp(x)	Returns the value of e raised to the power x(e**x)
expm1(x)	Returns the value of e raised to the power a (x-1)
log(x[, b])	Returns the logarithmic value of a with base b
log1p(x)	Returns the natural logarithmic value of 1+x

log2(x)	Computes value of log a with base 2
log10(x)	Computes value of log a with base 10
pow(x, y)	Compute value of x raised to the power y (x**y)
sqrt(x)	Returns the square root of the number
acos(x)	Returns the arc cosine of value passed as argument
asin(x)	Returns the arc sine of value passed as argument
atan(x)	Returns the arc tangent of value passed as argument

atan2(y, x)	Returns atan(y / x)
cos(x)	Returns the cosine of value passed as argument
hypot(x, y)	Returns the hypotenuse of the values passed in arguments
sin(x)	Returns the sine of value passed as argument
tan(x)	Returns the tangent of the value passed as argument
degrees(x)	Convert argument value from radians to degrees
radians(x)	Convert argument value from degrees to radians

<u>acosh(x)</u>	Returns the inverse hyperbolic cosine of value passed as argument
<u>asinh(x)</u>	Returns the inverse hyperbolic sine of value passed as argument
<u>atanh(x)</u>	Returns the inverse hyperbolic tangent of value passed as argument
<u>cosh(x)</u>	Returns the hyperbolic cosine of value passed as argument
<u>sinh(x)</u>	Returns the hyperbolic sine of value passed as argument
<u>tanh(x)</u>	Returns the hyperbolic tangent of value passed as argument
erf(x)	Returns the error function at x
erfc(x)	Returns the complementary error function at x
<u>gamma(x)</u>	Return the gamma function of the argument
lgamma(x)	Return the natural log of the absolute value of the gamma function

Python Random Module

- Python **Random module** is an in-built module of Python which is used to generate random numbers.
- These are pseudo-random numbers means these are not truly random.
- This module can be used to perform random actions such as generating random numbers, print random a value for a list or string, etc.
- **Example:** Printing a random value from a list

```
# import random
import random
# prints a random value from the list
list1 = [1, 2, 3, 4, 5, 6]
print(random.choice(list1))
```

- Random module creates pseudo-random numbers.
- Random numbers depend on the seeding value. For example, if the seeding value is 5 then the output of the below program will always be the same.
- The seed value is a base value used by a pseudo-random generator to produce random numbers.
- The random number or data generated by Python’s random module is not truly random; it is pseudo-random(it is PRNG), i.e., deterministic. The random module uses the seed value as a base to generate a random number.

Seed() in random

- The seed() method is used to initialize the random number generator.
- The random number generator needs a number to start with (a seed value), to be able to generate a random number.
- By default the random number generator uses the **current system time**.
- Use the seed() method to customize the start number of the random number generator.

Syntax:-

random.seed(*a*, *version*)

Parameter	Description
a	Optional. The seed value needed to generate a random number. If it is an integer it is used directly, if not it has to be converted into an integer. Default value is None, and if None, the generator uses the current system time.
version	An integer specifying how to convert the a parameter into a integer. Default value is 2

```
import random
random.seed(5)
print(random.random())
print(random.random())
```

Creating Random Integers

- **random.randint()** method is used to generate random integers between the given range.
- **Syntax :**
- randint(start, end)

- # import random module
- import random
- # Generates a random number between
- # a given positive range
- r1 = random.randint(5, 15)
- print("Random number between 5 and 15 is " , (r1))
- # Generates a random number between
- # two given negative range
- r2 = random.randint(-10, -2)
- print("Random number between -10 and -2 is " , (r2))

Creating Random Floats

- random.random() method is used to generate random floats between 0.0 to 1.
- **Syntax:**
- random.random()
- **Example:-**
- # import random
- from random import random
- # Prints random item
- print(random())

Selecting Random Elements

- random.choice() function is used to return a random item from a list, tuple, or string.
- **Syntax:**
- random.choice(sequence)
- **Example:-**

```
import random
# prints a random value from the list
list1 = [1, 2, 3, 4, 5, 6]
print(random.choice(list1))
# prints a random item from the string
string = "geeks"
print(random.choice(string))
# prints a random item from the tuple
tuple1 = (1, 2, 3, 4, 5)
print(random.choice(tuple1))
```

Shuffling List

- random.shuffle() method is used to shuffle a sequence (list). Shuffling means changing the position of the elements of the sequence. Here, the shuffling operation is inplace.
- **Syntax:**
- random.shuffle(sequence, function)

```
# import the random module
import random
sample_list = [1, 2, 3, 4, 5]
print("Original list : ")
print(sample_list)
# first shuffle
random.shuffle(sample_list)
print("\nAfter the first shuffle : ")
print(sample_list)
# second shuffle
random.shuffle(sample_list)
print("\nAfter the second shuffle : ")
print(sample_list)
```

List of all the functions in Random Module

Function Name	**Description**
seed()	Initialize the random number generator
getstate()	Returns an object with the current internal state of the random number generator
setstate()	Used to restore the state of the random number generator back to the specified state
getrandbits()	Return an integer with a specified number of bits
randrange()	Returns a random number within the range
randint()	Returns a random integer within the range

choice()	Returns a random item from a list, tuple, or string
choices()	Returns multiple random elements from the list with replacement
sample()	Returns a particular length list of items chosen from the sequence
random()	Generate random floating numbers
uniform()	Return random floating number between two numbers both inclusive
triangular()	Return a random floating point number within a range with a bias towards one extreme

betavariate()	Return a random floating point number with beta distribution
expovariate()	Return a random floating point number with exponential distribution
gammavariate()	Return a random floating point number with gamma distribution
gauss()	Return a random floating point number with Gaussian distribution
lognormvariate()	Return a random floating point number with log-normal distribution
normalvariate()	Return a random floating point number with normal distribution
vonmisesvariate()	Return a random floating point number with von Mises distribution or circular normal distribution

paretovariate()	Return a random floating point number with Pareto distribution
weibullvariate()	Return a random floating point number with Weibull distribution

OS Module in Python

- Python OS module provides the facility to establish the interaction between the user and the operating system.
- It offers many useful OS functions that are used to perform OS-based tasks and get related information about operating system.
- The OS comes under Python's standard utility modules.
- This module offers a portable way of using operating system dependent functionality.
- The Python OS module lets us work with the files and directories.

- For example, it makes it possible for us to get the path of the directory we are working with, get the names of all the files and folders within a directory, make a new directory or remove an existing one and so on.

Directory related functions

- **Handling the Current Working Directory**
- Consider **Current Working Directory(CWD)** as a folder, where the Python is operating.
- Whenever the files are called only by their name, Python assumes that it starts in the CWD which means that name-only reference will be successful only if the file is in the Python's CWD.

1. os.getcwd():-

- It returns the current working directory(CWD) of the file.

Example:-

```
import os
print(os.getcwd())
```

Output:

```
C:\Users\Python\Desktop
```

2. os.chdir()

- The **os** module provides the **chdir()** function to change the current working directory.

Example:

```
Import os
os.chdir(“C:/Users/Dharam/Desktop/itm/BTech_1sem(AICSDA)_python programming1)
```

- **Creating a Directory**
- There are different methods available in the OS module for creating a directory.
- os.mkdir()

1. os.mkdir()

- os.mkdir() method in Python is used to create a directory named path with the specified numeric mode.
- This method raises FileExistsError if the directory to be created already exists.

Example:-

```
import os
os.mkdir("d:\\newdir")
```

- **Listing out Files and Directories with Python**
- **os.listdir()** method in Python is used to get the list of all files and directories in the specified directory.
- If we don’t specify any directory, then the list of files and directories in the current working directory will be returned.

Example:-

```
import os
os.listdir("C:/")
```

- **Deleting Directory or Files using Python**
- OS module proves different methods for removing directories and files in Python.
- **os.remove()**
- os.remove() method in Python is used to remove or delete a file path.
- This method can not remove or delete a directory. If the specified path is a directory then OSError will be raised by the method.

Example:-

```
import os
os.remove("C:/python/file1.txt")
```

- **Using os.rmdir()**
- os.rmdir() method in Python is used to remove or delete an empty directory. OSError will be raised if the specified path is not an empty directory.

Example:-

```
Import os
os.rmdir("C:/python")
```

- **Commonly Used Functions**
- **os.name()**
- This function provides the name of the operating system module that it imports.
- Currently, it registers 'posix', 'nt', 'os2', 'ce', 'java' and 'riscos'.
- **Example**

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

- **os.path.getsize():**
- In this method, python will give us the size of the file in bytes.
- To use this method we need to pass the name of the file as a parameter.
- **Example:-**

```
import os
size=os.path.getsize("C:/Users/Dharam/Desktop/itm/BTech_1sem(AICSDA)_python programming1/FINAL PPT MATERIAL/FINAL PPT MATERIAL/att.txt")
print("Size of the file is", size," bytes.")
```

Python sys Module

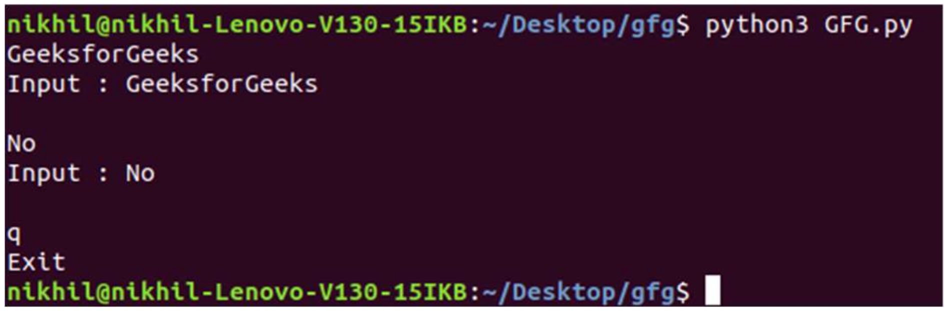
- The **sys module** in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment.
- It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.
- **sys.version():-**
- **sys.version** is used which returns a string containing the version of Python Interpreter with some additional information.
- This shows how the sys module interacts with the interpreter.
- **Example:-**

```
import sys
print(sys.version)
```

- **Input and Output using sys**
- The sys modules provide variables for better control over input or output.
- We can even redirect the input and output to other devices. This can be done using three variables –
- **Stdin:-**This is the *file-handle* that a user program reads to get information from the user. We give input to the standard input (**stdin**).
- **Stdout:-** The user program writes normal information to this file-handle. The output is returned via the Standard output (**stdout**).
- **Stderr:-** The user program writes error information to this file-handle. Errors are returned via the Standard error (**stderr**)

- **stdin:** It can be used to get input from the command line directly. It used is for standard input.
- **It internally calls the input() method**. It, also, automatically adds '\n' after each sentence.
- This is similar to a file, where you can open and close it, just like any other file.
- **Example:**

```
import sys
for line in sys.stdin:
        if 'q' == line.rstrip():
                break
        print('Input : {line}')
print("Exit")
```



```
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$ python3 GFG.py
GeeksforGeeks
Input : GeeksforGeeks

No
Input : No

q
Exit
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$
```

- **stdout:** A built-in file object that is analogous to the interpreter’s standard output stream in Python.
- stdout is used to display output directly to the screen console.
- Output can be of any form, it can be output from a print statement, an expression statement, and even a prompt direct for input.
- By default, streams are in text mode.
- In fact, wherever a print function is called within the code, it is first written to sys.stdout and then finally on to the screen.

- **Example:-**

import sys

sys.stdout.**write(‘Hello’)**

- **<u>stderr:</u>**
- Whenever an exception occurs in Python it is written to sys.stderr.
- This is similar to sys.stdout because it also prints directly to the Console. But the difference is that it *only* prints **Exceptions** and **Error messages**. (Which is why it is called **Standard Error**).
- **Example:-**

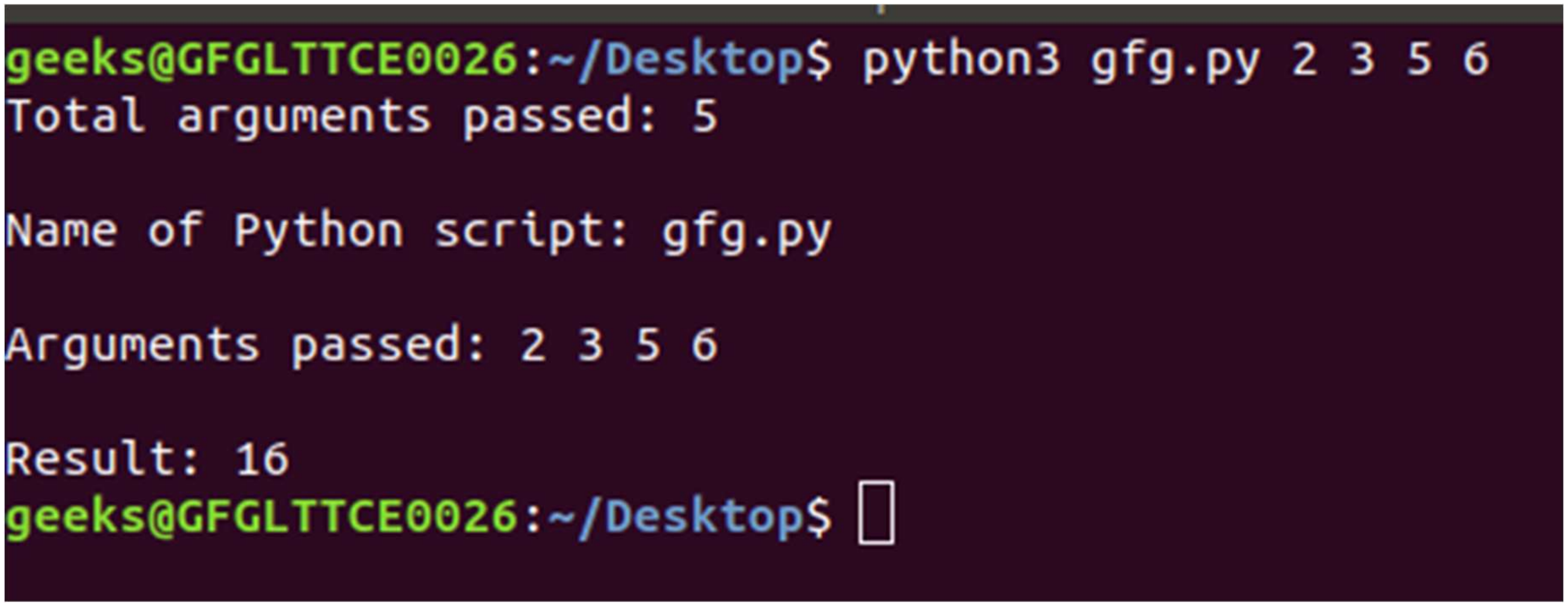
```
import sys
def print_to_stderr(*a):
        # Here a is the array holding the objects
        # passed as the argument of the function
        print(*a, file = sys.stderr)
print_to_stderr("Hello World")
```

Command Line Arguments

- Command-line arguments are those which are passed during the calling of the program along with the calling statement.
- To achieve this using the sys module, the sys module provides a variable called **sys.argv.** It’s main purpose are:
  - **It is a list of command-line arguments.**
  - **len(sys.argv) provides the number of command-line arguments.**
  - **sys.argv[0] is the name of the current Python script.**

Consider a program for adding numbers and the numbers are passed along with the calling statement.

```
import sys
# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)
# Arguments passed
print("\nName of Python script:", sys.argv[0]) #position of python file is  0
print("\nArguments passed:", end = " ")
for i in range(1, n): #n=5
        print(sys.argv[i], end = " ")
# Addition of numbers
Sum = 0
for i in range(1, n):
        Sum += int(sys.argv[i])
print("\n\nResult:", Sum)
```



```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py 2 3 5 6
Total arguments passed: 5

Name of Python script: gfg.py

Arguments passed: 2 3 5 6

Result: 16
geeks@GFGLTTCE0026:~/Desktop$ 
```

- **Exiting the Program**
- **sys.exit([arg])** can be used to exit the program.
- The optional argument arg can be an integer giving the exit or another type of object.
- If it is an integer, **zero is considered “successful termination”.**
- A string can also be passed to the sys.exit() method.

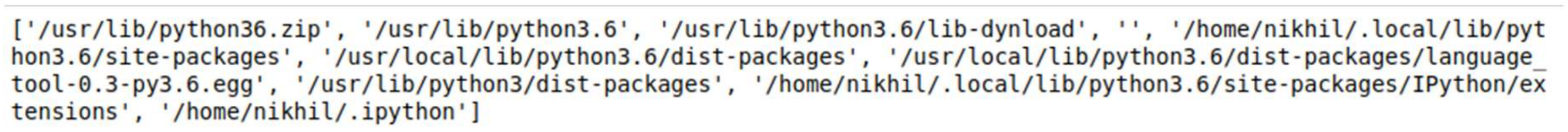
```
import sys
age = 17
if age < 18:
        # exits the program
        sys.exit("Age less than 18")
else:
        print("Age is not less than 18")
```

- **Working with Modules**
- **sys.path** is a built-in variable within the sys module that returns the list of directories that the interpreter will search for the required module.
- When a module is imported within a Python file, the interpreter first searches for the specified module among its built-in modules.
- If not found it looks through the list of directories defined by **sys.path**.
- sys.path is an ordinary list and can be manipulated.

- **Example:-**

```
import sys
print(sys.path)
```

Output:-

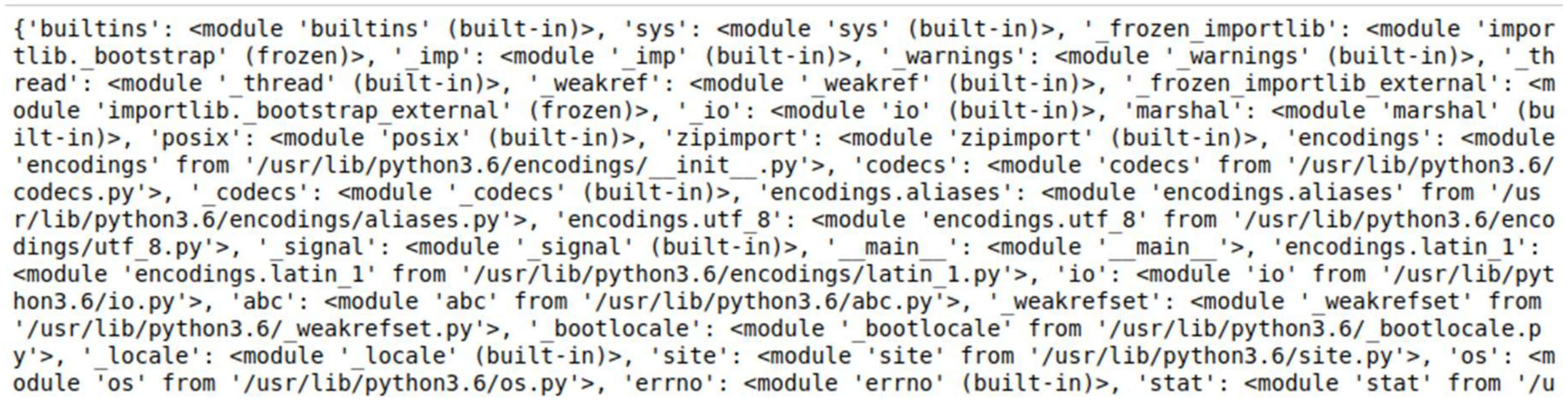


```
['/usr/lib/python36.zip', '/usr/lib/python3.6', '/usr/lib/python3.6/lib-dynload', '', '/home/nikhil/.local/lib/pyt
hon3.6/site-packages', '/usr/local/lib/python3.6/dist-packages', '/usr/local/lib/python3.6/dist-packages/language_
tool-0.3-py3.6.egg', '/usr/lib/python3/dist-packages', '/home/nikhil/.local/lib/python3.6/site-packages/IPython/ex
tensions', '/home/nikhil/.ipython']
```

- **sys.modules** return the name of the Python modules that the current shell has imported.
- **Example:-**

import sys

print(sys.modules)



```
{'builtins': <module 'builtins' (built-in)>, 'sys': <module 'sys' (built-in)>, '_frozen_importlib': <module 'impor
tlib._bootstrap' (frozen)>, '_imp': <module '_imp' (built-in)>, '_warnings': <module '_warnings' (built-in)>, '_th
read': <module '_thread' (built-in)>, '_weakref': <module '_weakref' (built-in)>, '_frozen_importlib_external': <m
odule 'importlib._bootstrap_external' (frozen)>, '_io': <module 'io' (built-in)>, 'marshal': <module 'marshal' (bu
ilt-in)>, 'posix': <module 'posix' (built-in)>, 'zipimport': <module 'zipimport' (built-in)>, 'encodings': <module
'encodings' from '/usr/lib/python3.6/encodings/__init__.py'>, 'codecs': <module 'codecs' from '/usr/lib/python3.6/
codecs.py'>, '_codecs': <module '_codecs' (built-in)>, 'encodings.aliases': <module 'encodings.aliases' from '/us
r/lib/python3.6/encodings/aliases.py'>, 'encodings.utf_8': <module 'encodings.utf_8' from '/usr/lib/python3.6/enco
dings/utf_8.py'>, '_signal': <module '_signal' (built-in)>, '__main__': <module '__main__'>, 'encodings.latin_1':
<module 'encodings.latin_1' from '/usr/lib/python3.6/encodings/latin_1.py'>, 'io': <module 'io' from '/usr/lib/pyt
hon3.6/io.py'>, 'abc': <module 'abc' from '/usr/lib/python3.6/abc.py'>, '_weakrefset': <module '_weakrefset' from
'/usr/lib/python3.6/_weakrefset.py'>, '_bootlocale': <module '_bootlocale' from '/usr/lib/python3.6/_bootlocale.p
y'>, '_locale': <module '_locale' (built-in)>, 'site': <module 'site' from '/usr/lib/python3.6/site.py'>, 'os': <m
odule 'os' from '/usr/lib/python3.6/os.py'>, 'errno': <module 'errno' (built-in)>, 'stat': <module 'stat' from '/u
```

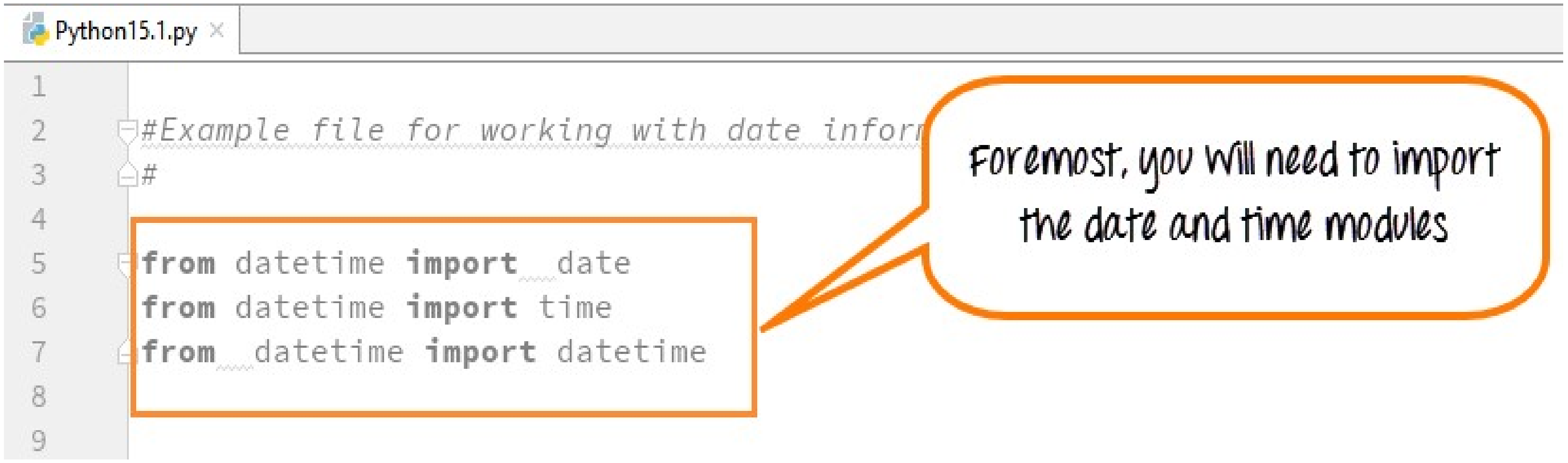
- **Reference Count**
- **sys.getrefcount()** method is used to get the reference count for any given object. This value is used by Python as when this value becomes 0, the memory for that particular value is deleted.
- **Example:-**

```
import sys
a = ‘Hello'
print(sys.getrefcount(a))
```

Python datetime module

- In Python, date and time are not a data type of their own, but a module named **datetime** can be imported to work with the date as well as time.
- **Python Datetime module** comes built into Python, so there is no need to install it externally.
- Python Datetime module supplies classes to work with date and time.
- These classes provide a number of functions to deal with dates, times and time intervals.
- Date and datetime are an object in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

- The DateTime module is categorized into 6 main classes –
- **date** – An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Its attributes are year, month and day.
- **time** – An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Its attributes are hour, minute, second, microsecond, and tzinfo.
- **datetime** – Its a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo.
- **timedelta** – A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
- **tzinfo** – It provides time zone information objects.
- **timezone** – A class that implements the tzinfo abstract base class as a fixed offset from the UTC (New in version 3.2).



```

#Example file for working with date inforr
#

from datetime import  date
from datetime import time
from  datetime import datetime


```

Date Class

- **Get Current Date**
- To return the current local date today() function of date class is used. today() function comes with several attributes (year, month and day). These can be printed individually.
- **Example:-**

```
from datetime import date

# calling the today function of date class

today = date.today() #print today’s date

print("Today's date is", today)
```

- **Get Today’s Year, Month, and Date**
- We can get the year, month, and date attributes from the date object using the year, month and date attribute of the date class.
- **Example:-**

```
from datetime import date
# date object of today's date
today = date.today()
print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

Datetime Class:-

- **Get date from Timestamp**
- We can create date objects from timestamps using the fromtimestamp() method.
- The timestamp is the number of seconds from 1st January 1970 at UTC to a particular date.
- **Example:-**

```
from datetime import datetime
# Getting Datetime from timestamp
date_time = datetime.fromtimestamp(1887639468)
print("Datetime from timestamp:", date_time)
```

- **Convert Date to String**
- We can convert date object to a string representation using two functions isoformat() and strftime().
- **Example:-**

```
from datetime import date
# calling the today function of date class
today = date.today()
# Converting the date to the string
Str = date.isoformat(today)
print("String Representation", Str)
print(type(Str))
```

- The DateTime class contains information on both date and time. Like a date object, datetime assumes the current Gregorian calendar extended in both directions; like a time object, datetime assumes there are exactly 3600*24 seconds in every day.
- **Get year, month, hour, minute, and timestamp**
- After creating a DateTime object, its attributes can also be printed separately.
- **Example:-**

```
from datetime import datetime
a = datetime(1999, 12, 12, 12, 12, 12)
print("year =", a.year)
print("month =", a.month)
print("hour =", a.hour)
print("minute =", a.minute)
print("timestamp =", a.timestamp())
```

- **Current date and time**
- You can print the current date and time using the Datetime.now() function. now() function returns the current local date and time.
- **Example:-**

```
from datetime import datetime
# Calling now() function
today = datetime.now()
print("Current date and time is", today)
```

- **Convert Python Datetime to String**
- We can convert Datetime to string in Python using the datetime.strftime and datetime.isoformat methods.
- **Example:-**

```
from datetime import datetime as dt
# Getting current date and time
now = dt.now()
string = dt.isoformat(now)
print(string)
print(type(string))
```

Time class

- The time class creates the time object which represents local time, independent of any day.
- **Get hours, minutes, seconds, and microseconds**
- After creating a time object, its attributes can also be printed separately.
- **Example:-**

```
from datetime import time
Time = time(11, 34, 56)
print("hour =", Time.hour)
print("minute =", Time.minute)
print("second =", Time.second)
print("microsecond =", Time.microsecond)
```

- **Convert Time object to String**
- We can convert time object to string using the isoformat() method.
- **Example:-**

```
from datetime import time
# Creating Time object
Time = time(12,24,36,1212)
# Converting Time object to string
Str = Time.isoformat()
print("String Representation:", Str)
print(type(Str))
```

Timedelta class

- Python timedelta class is used for calculating differences in dates and also can be used for date manipulations in Python.
- It is one of the easiest ways to perform date manipulations.
- **Difference between two date and times**
- Date and Time difference can also be found using this class.

Example:-

```
# Timedelta function demonstration
from datetime import datetime, timedelta
print("today is :-"+(str(datetime.now())))
print("Date after one year
is:"+(str(datetime.now()+timedelta(days=365))))
print("Date after one week and 5 days is :-
"+(str(datetime.now()+timedelta(weeks=1, days=5))))
```

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