PySpark

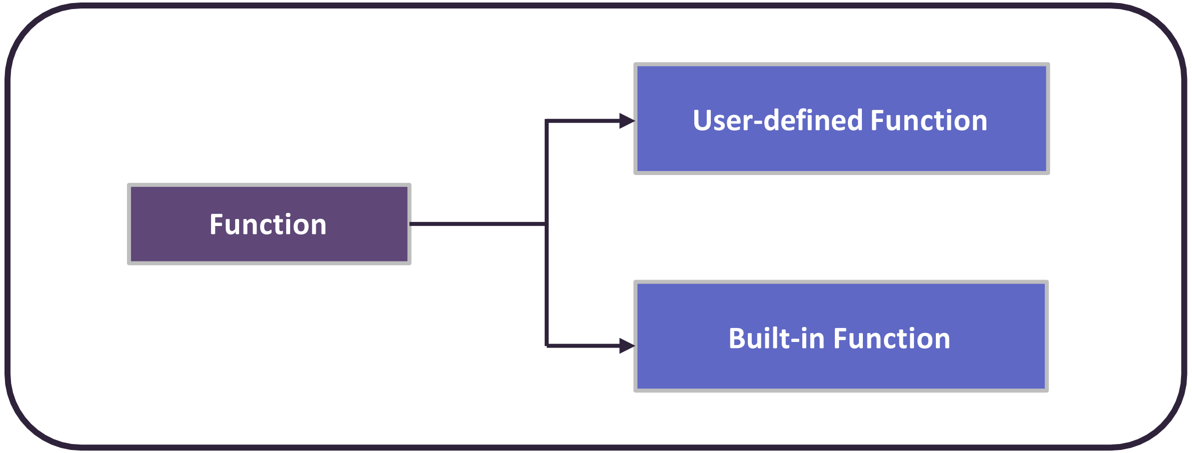
Python for Spark: Functional and Object-oriented Model

Agenda

Functions in **Python**

What are Functions?

A function is a block of organized, reusable sets of instructions that is used to perform some related actions



Types of Functions

User-defined Function

Built-in Function

Syntax

```
def func_name (arg1, arg2, arg3, ….):
         statements…
         return [expression]
```

Example

```
def add ( a, b )
  sum = a + b
  return sum
```

Types of Functions

User-defined Function

Built-in Function

- **abs():** Returns the absolute value of a number
- **all():** Returns True if all items in an iterable object are true
- **any():** Returns True if any item in an iterable object is true
- **ascii():** Returns a readable version of an object
- **bin():** Returns the binary version of a number
- **bool():** Returns the Boolean value of the specified object

Function Call

Call by Value

Call by Reference

What is Call by Value?

- The call by value method of passing arguments to a function copies the actual value of an argument into the formal parameter of the function
- Changes made to the parameter inside the function have no effect on the argument

Function Call

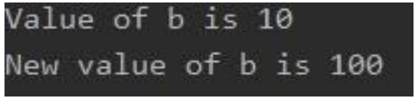
Call by Value

Call by Reference

Syntax

```
>>>a=10
>>>def ChangeIt(b):
…           print(“value of b is”, b)
…           b=100
…           print(“New value of b is”, b)
…
>>>ChangeIt(a)
```

Output



```
Value of b is 10
New value of b is 100
```

Function Call

Call by Value

Call by Reference

What is Call by Reference?

- The call by reference method of passing arguments to a function copies the address of an argument into the formal parameter
- Inside the function, the address is used to access the actual argument used in the call

Function Call

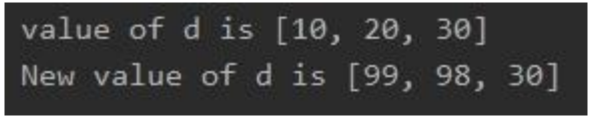
Call by Value

Call by Reference

Syntax

```
>>>c=[10,20,30]
>>>def ChangeThem(d):
…           print(“value of d is”, d)
…           d[0]=99
…           d[1]=98
…           print(“New value of d is”, d)
…
>>>ChangeThem(c)
```

Output

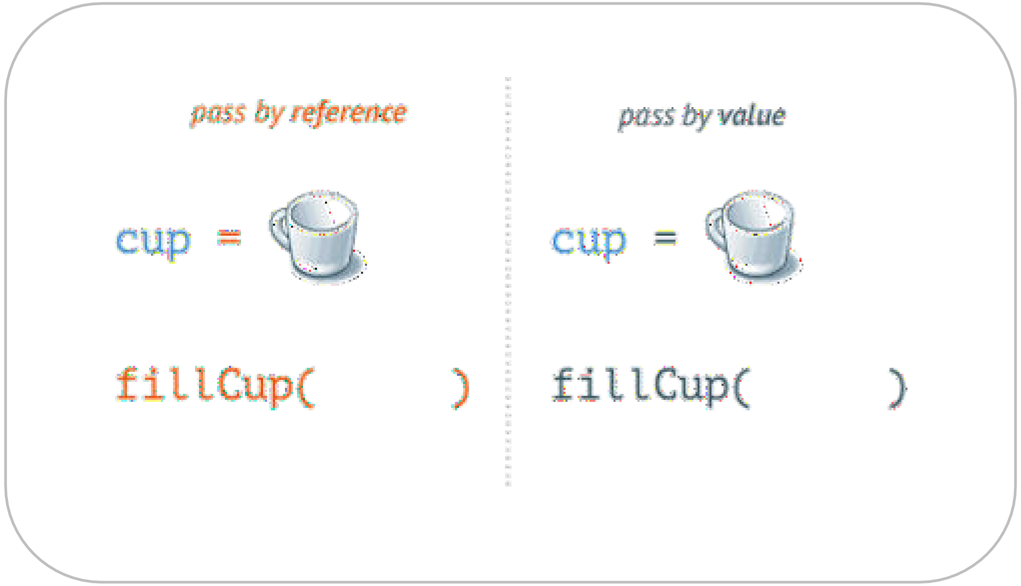


```
value of d is [10, 20, 30]
New value of d is [99, 98, 30]
```

Function Call

Call by Value

Call by Reference



Eventual goal of filling up the cup is fulfilled because of the parameter value being copied from the cup which reflects in the function

Variables in **Python**

Variables in Python

Recap:

- The name of a variable cannot start with a number. It should start with either an alphabet or the underscore character
- Variable names are always case sensitive and can contain alphanumeric characters and the underscore character
- Reserved words cannot be used as variable names
- Python variables are always assigned using the equal to sign followed by the value of the variable

Multiple Assignment

```
a = b = c = 5
```

Python is a type-inferred language, i.e., it automatically detects the type of the assigned variable

Example

```
test1=”String”
type(test1)
```

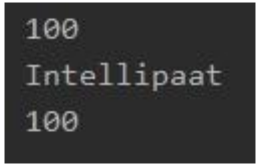
Local Variables

- A variable that is declared inside a Python function or a module can only be used in that specific instance, and it is called a **local variable**
- Python interpreter will not recognize the variable outside that specific function or module and will throw an error if that variable is not declared outside of the scope of the function

Example

```
a=100
print (f)
def some_function()
f = ‘Intellipaat’
print(f)
some_function()
print(f)
```

Output



```
100
Intellipaat
100
```

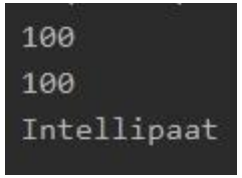
Global Variables

- A **global variable** in Python is a variable that can be used globally anywhere in the program
- It can be used in **any** function or module, and even outside the functions, **without** having to re-declare it

Example

```
a = 100
print (a)
def som_function():
global a
print (a)
a = ‘Intellipaat’
some_function()
print (a)
```

Output



```
100
100
Intellipaat
```

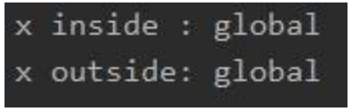
Global Variables

- Here’s another example:

Example

```
x = "global“

def foo():
    print("x inside :", x)

foo()
print("x outside:", x)
```

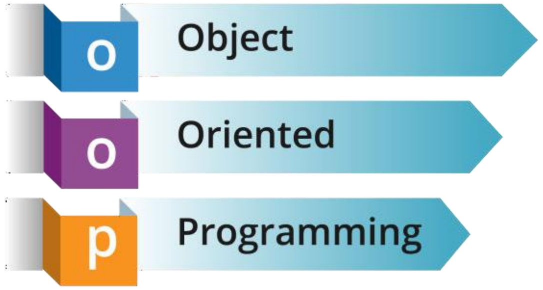
Output



```
x inside : global
x outside: global
```

OOP Concepts in **Python**

What is OOP?



Example:

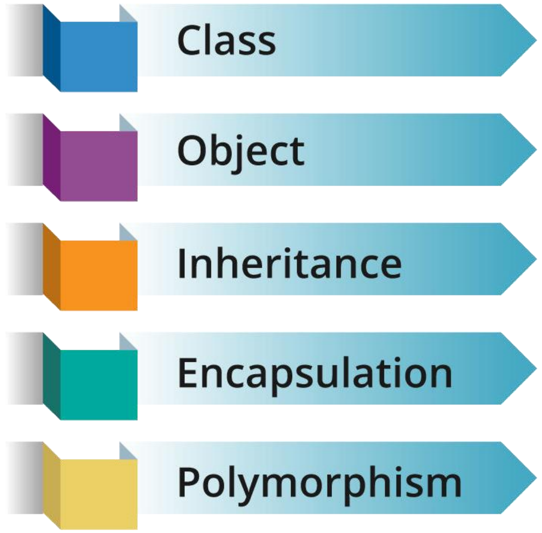
Parrot

Attribute: Name, Age, and Colour

Behavior: Singing and Dancing

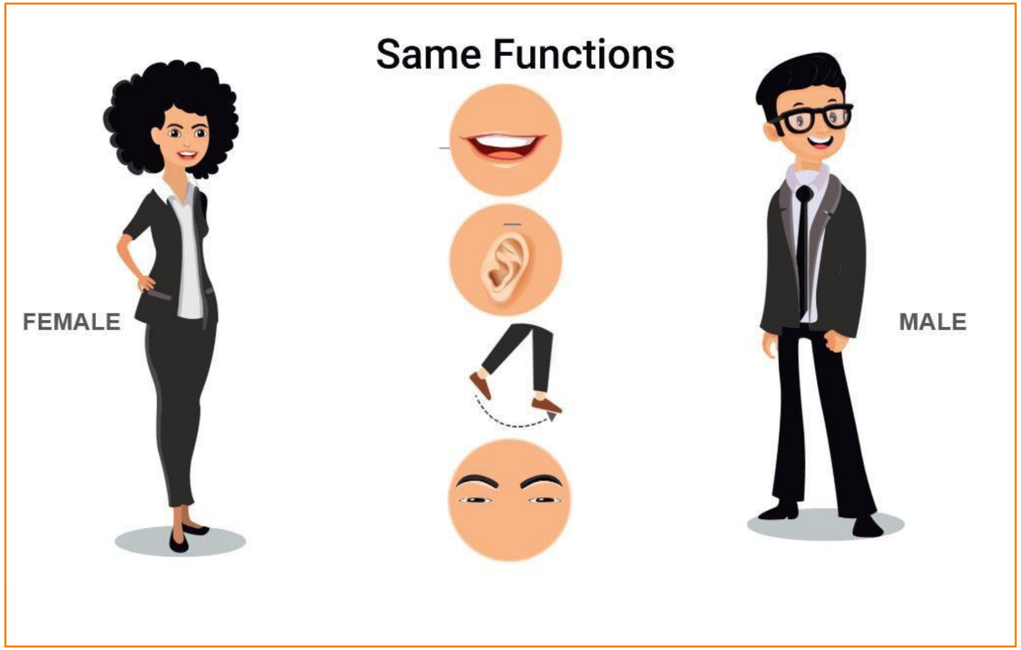
What is OOP?

Basic Principles of OOPS:



Real-life Example

Every human being is classified into:



Real-life Example

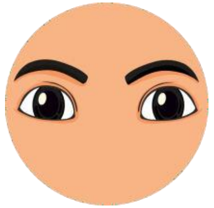
Every human has the following body parts:

NOSE

HAND

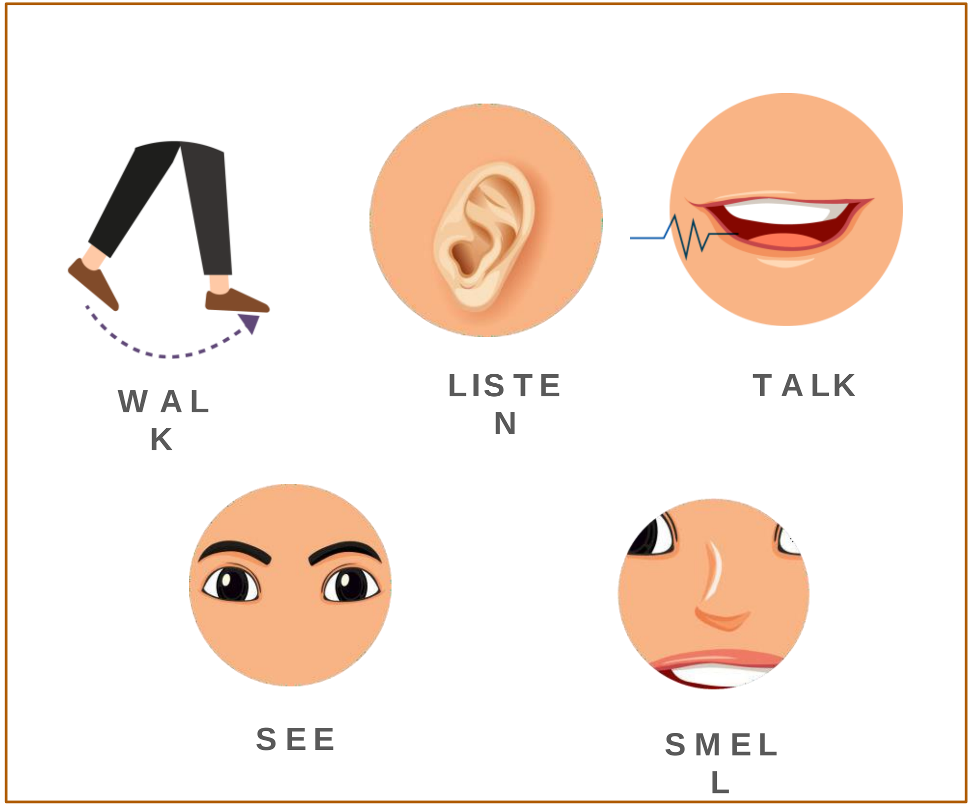
LEGS

HEAR
T


EYES

Real-life Example

Common human functions:

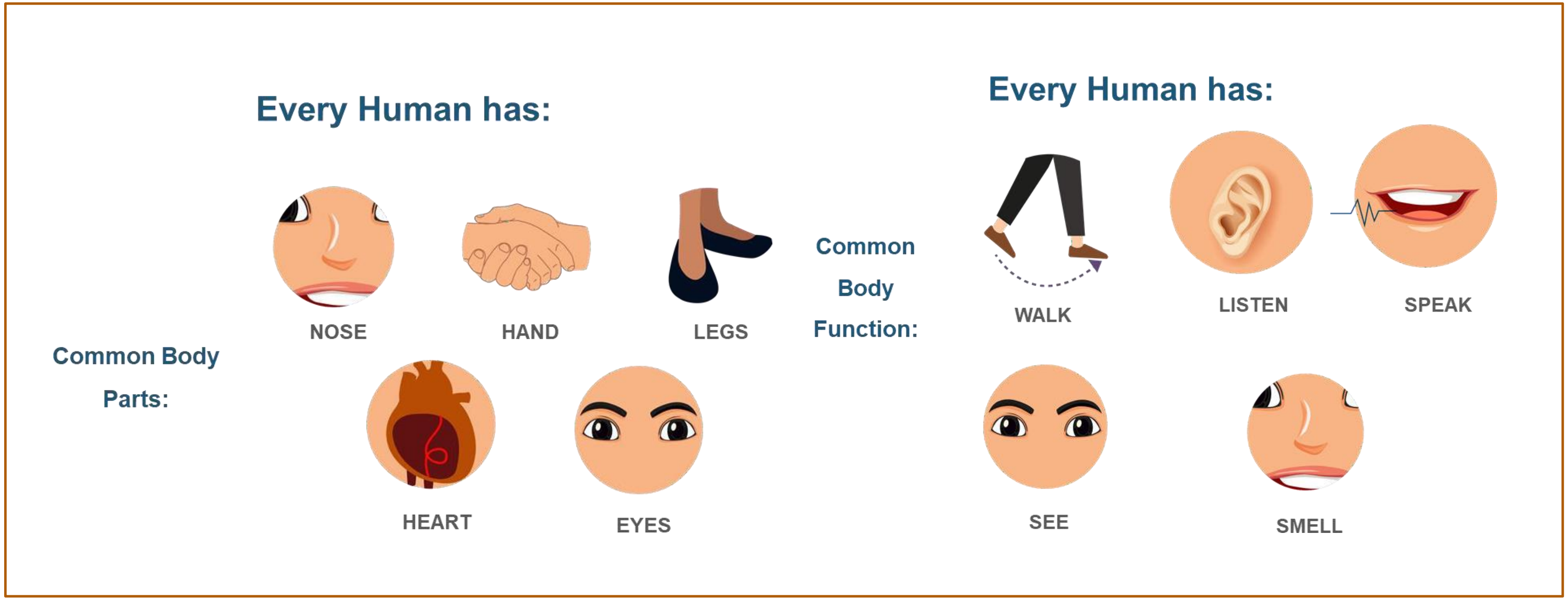


Real-life Example

Consider ‘Human Being’ as a class:

Real-life Example

Common body features and functions are class attributes:



Real-life Example

Let’s look at inheritance now:

MALE and **FEMALE** are inherited from the class ‘**Human Being**’!

FEMALE

MALE

Real-life Example

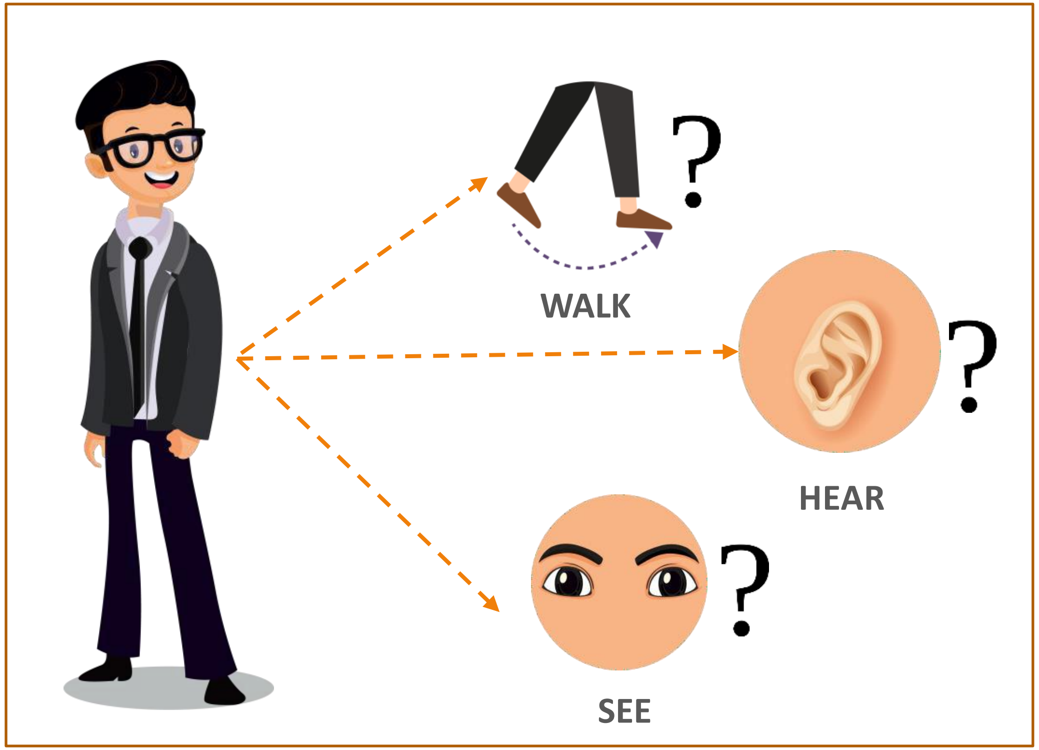
‘**Name**’ and ‘**Age**’ are the objects of the class **MALE**

- Objects have a physical existence
- Class is just a logical definition

Class: MALE
Name: Victor
Age: 24

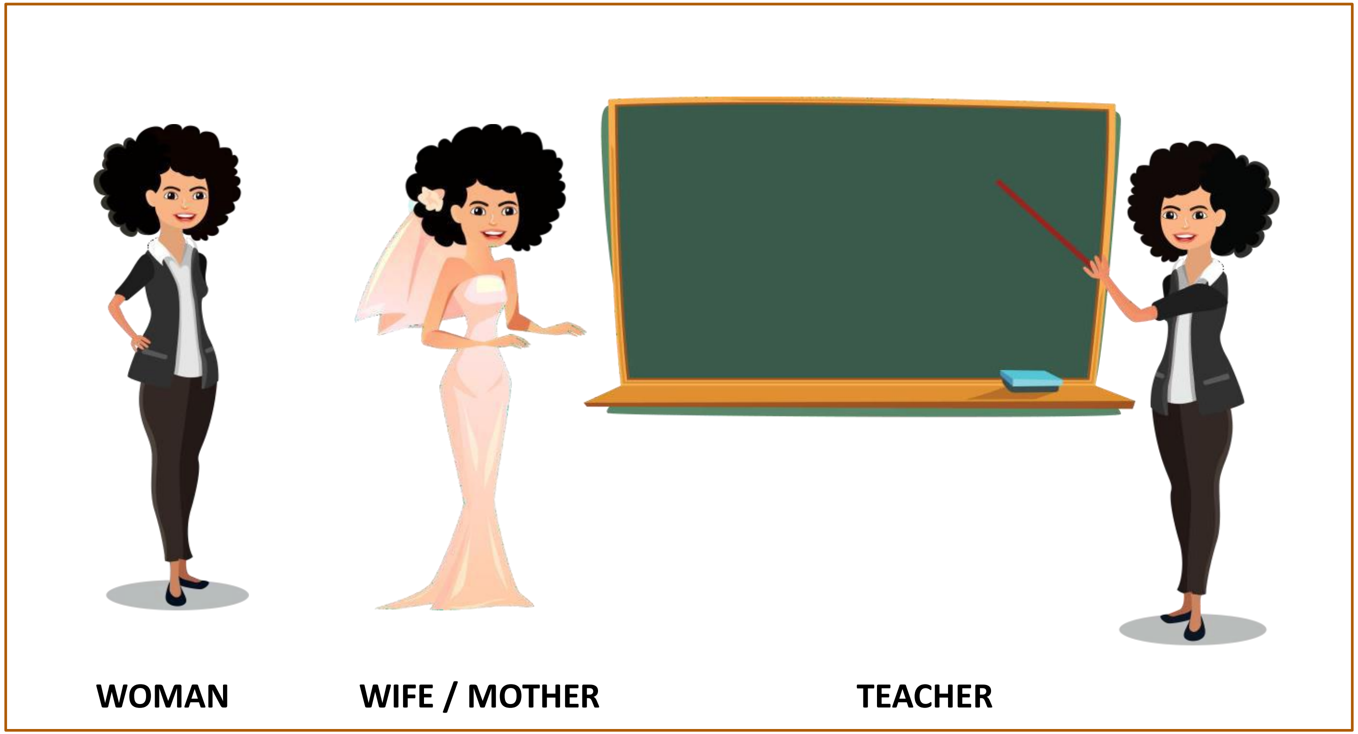
Real-life Example

You don’t know the details of how you walk, listen, or see, i.e., it is hidden!



Real-life Example

‘She’ can be a woman, wife, mother, and a teacher at the same time



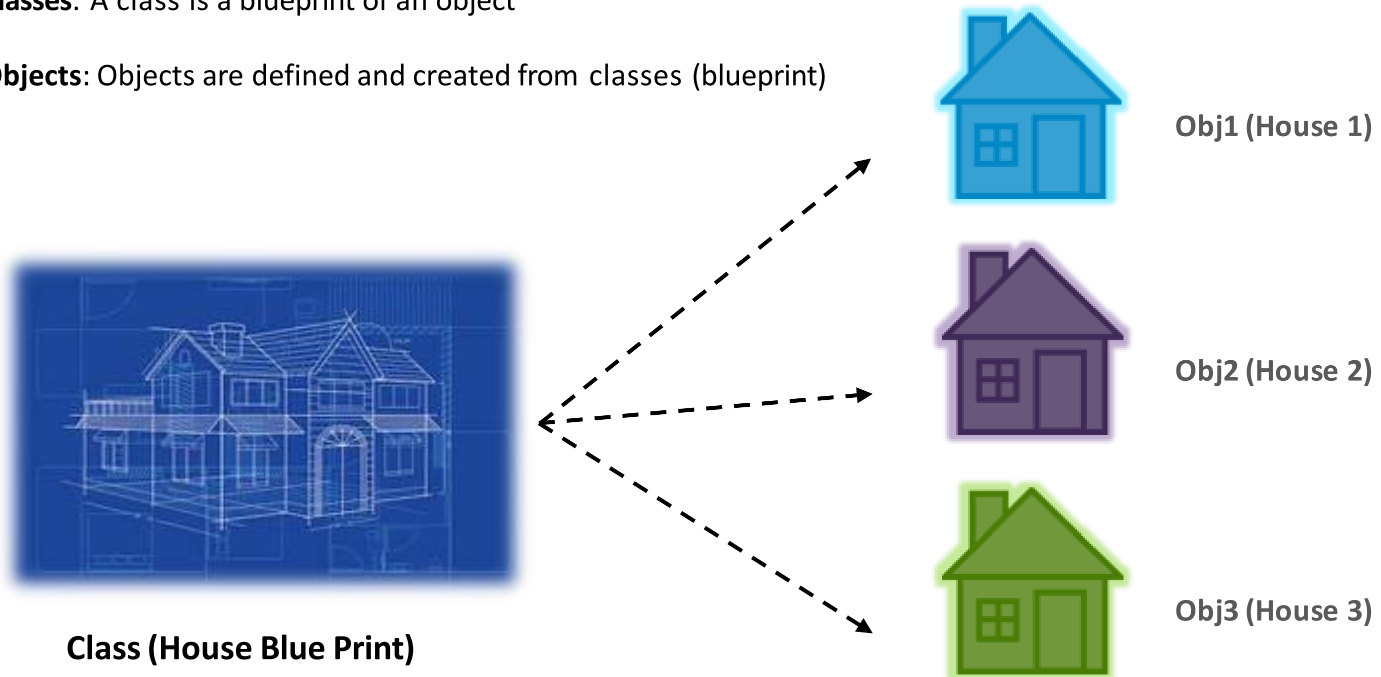
OOPS: Classes and Objects

- What are classes and objects?
- How to create a class in Python?
- How to create objects in Python?
- How to access class members?

OOPS: Classes and Objects

What are classes and objects?

- **Classes**: A class is a blueprint of an object
- **Objects**: Objects are defined and created from classes (blueprint)



OOPS: Classes and Objects

What are classes and objects?

- An object is the basic unit of the object-oriented programming
- An object represents an instance of a class
- There can be more than one instance of an object
- Each instance of an object can hold its own relevant data
- Objects with similar properties and methods are grouped together to form a class

OOPS: Classes and Objects

How to create a class in Python?

Syntax

```
class NameOfClass:
           <statement-1>
           .
           .
           <statement-2>
```

Example

```
class ClassName:
        variable= “I am a class Attribute”
        def function(self):
                print(“I am from inside the class”)
```

OOPS: Classes and Objects

How to create an object in Python?

Syntax

```
<obj-name> = NameOfClass()
```

Example

```
obj1 = ClassName()
```

Here, **obj1** is an object of the class **ClassName**

OOPS: Classes and Objects

How to access class members?

Example

```
obj1 = ClassName()
obj2 = ClassName()
#Creating new instance attribute for obj2
obj2.variable = “I was just created”
print(obj1.variable)
print(obj2.variable)
print(ClassName.variable)
Obj1.function()
```

- Here, **obj1** and **obj2** are the objects of the class **ClassName**
- To access the members of a Python class, we use the dot operator

OOPS: Classes and Objects

Inheritance in Python

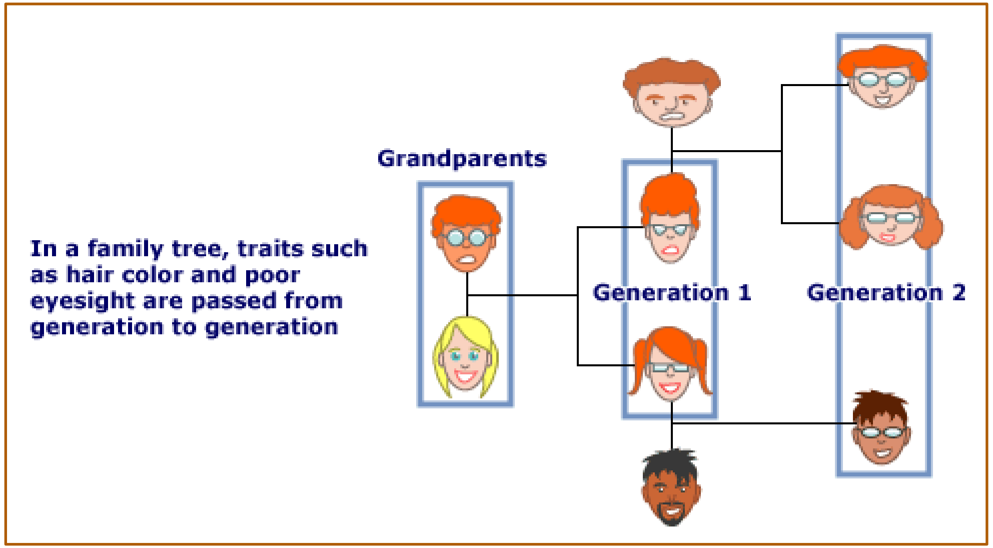
- What is Inheritance?
- Real-life example of Inheritance
- Different types of Inheritance in Python
- Overriding vs Overloading

OOPS: Classes and Objects

What is Inheritance in Python?

‘One class acquiring the property of another class’

E.g.: You have inherited some qualities from your parents


Grandparents
In a family tree, traits such as hair color and poor eyesight are passed from generation to generation
Generation 1
Generation 2

OOPS: Classes and Objects

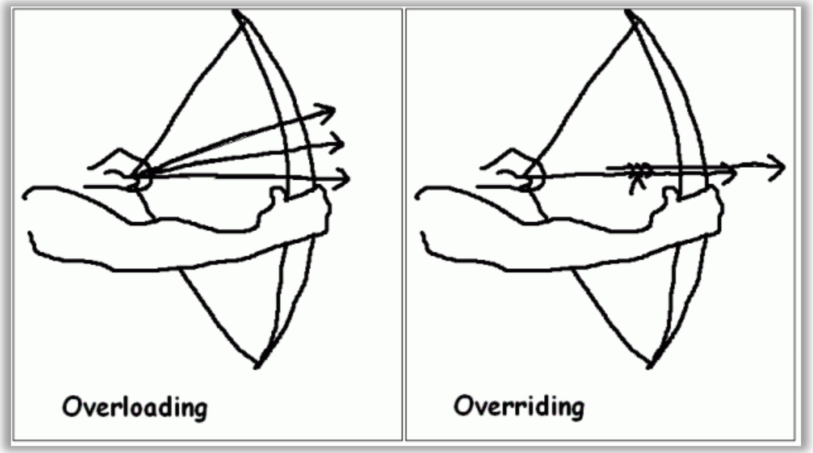
Creating a class in Python

```
class ClassName:
          <statement-1>
          .
          .
          <statement-2>
```

OOPS: Classes and Objects

Overriding vs Overloading

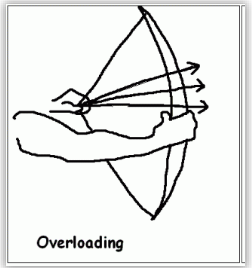
‘Developers sometimes get confused between them’



OOPS: Classes and Objects

Overloading a function

'Same function with different parameters'



✕

Why Overload a Function?

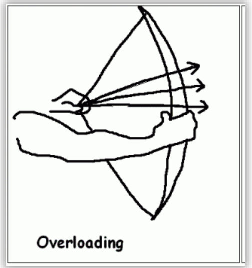
```
def add(a,b):
      return a+b
def add(a,b,c):
      return a+b+c
add(2,3)
```

```
TypeError: add() missing 1
required positional argument: 'c'
```

OOPS: Classes and Objects

Overloading a function

‘Used to call a parent method from the class’



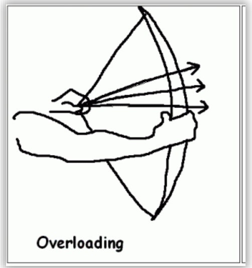
How to Overload a Function?

```
def add(instanceOf,*args):
      if instanceOf=='int':
              result=0
      if instanceOf=='str':
              result=''
      for i in args:
              result+=i
      return result
add('int',3,4,5)
```

OOPS: Classes and Objects

Overriding a function

‘A subclass may change the functionality of a Python method in the superclass’



How to Overload a Function?

```
class A:
          def sayhi(self):
                    print("I'm in A")
class B(A):
          def sayhi(self):
                     print("I'm in B")
bobj=B()
bobj.sayhi()
```

OOPS: Classes and Objects

Encapsulation in Python

- What is Encapsulation?
- Real-life example of Encapsulation
- How to access a private method?
- How to access a private variable?

OOPS: Classes and Objects

What is Encapsulation?

Abstraction + Data Hiding

Abstraction is showing the essential features and hiding the non-essential features to the user

Example:

While writing a mail, you don’t know how things are happening in the backend

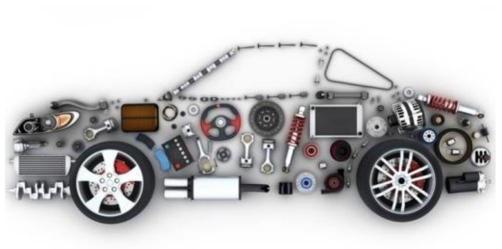
OOPS: Classes and Objects

What is Encapsulation?

Abstraction + Data Hiding

Wrapping up of data into a single unit is called Encapsulation

Example:



Multiple parts of a car encapsulate themselves together to form a single object, i.e., the Car

OOPS: Classes and Objects

What is Encapsulation?

Convention:

A class variable that should not directly be accessed (private) should be prefixed with an underscore

Encapsulating a Function

```
class Encap(object):
   def__init__(self):
      self.a = 123
      self._b = 123
      self.__c = 123

obj = Encap()
print(obj.a)
print(obj._b)
print(obj.__c)
```

❌

```
AttributeError: 'Encap' object
has no attribute '__c'
```

So, what's with the underscore and the error?

OOPS: Classes and Objects

How to access a private method?

A private method can be called using redcar._Car__updateSoftware()

Example

```
class Car:

    def __init__(self):
        self.__updateSoftware()

    def drive(self):
        print ('driving')

    def __updateSoftware(self):
        print ('updating software')

redcar  = Car()
redcar.drive()
redcar._Car__updateSoftware()
```

OOPS: Classes and Objects

How to access a private variable?

To change the value of a private variable, a setter method is used

Example

```
 def setMaxSpeed(self,speed):
        self.__maxspeed = speed


redcar = Car()
redcar.drive()
redcar.__maxspeed = 10  # will not
change variable because its private
redcar.setMaxSpeed(320)
redcar.drive()
```

OOPS: Classes and Objects

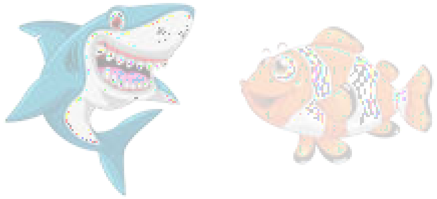
What is Polymorphism?

‘Functions with same name but functioning in different ways’

For Example:

OOPS: Classes and Objects

Polymorphism with a Function



Example

```
def in_the_pacific(fish):
    fish.swim()

sammy = Shark()

casey = Clownfish()

in_the_pacific(sammy)
in_the_pacific(casey)
```

Lambda Functions in **Python**

What is Lambda Function?

- The **lambda** keyword is used to define anonymous functions, i.e., functions without names. Python lambda functions are **not much different** from the regular functions that are defined using the **def** keyword

- Using the lambda function in certain situations makes it a bit easier and cleaner to write the code

Function

```
def multiply(x, y):
    return x * y
```

Lambda Function

```
r = lambda x, y: x * y
r(12, 3)   # call the lambda function
```

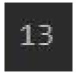
What is Lambda Function?

Power of Lambda: Anonymous Function Inside Another Function

Example

```
def myfunc(n):
  return lambda a : a + n
mySum = myfunc(3)
print(mySum(10))
```

Output

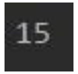


```
13
```

Example

```
x = lambda a : a + 10
print(x(5))
```

Output



```
15
```

What is Lambda Function?

Features of Lambda Functions

- Anonymous functions created using the lambda keyword can have any number of arguments, but they are syntactically restricted to just one expression, i.e., they can have only one expression
- Lambda function in Python can be used wherever a function object is required
- Lambda functions do not require any return statement; they always return a value obtained by evaluating the lambda expression in Python
- Python Lambda functions are widely used with some Python built-in functions such as map(), reduce(), etc.

Modules in **Python**

Modules in Python

What are modules?

- When we write a program in Python interpreter or Python shell and then exit from the shell, all the definitions that we had included in our program get lost. We can’t use those definitions again
- While working on a project that deals with various long programs, it’s better to just use a text editor and create scripts of the **py** extension
- Now if you are using the same function in different programs, you won’t have to define it again and again. You can just create a script containing that function and import that script in every program that makes use of that function
- **These scripts are called modules in Python**

Modules Visualized

Consider the following situation:

- Say, a Software Developer, Leon was given a task to develop a software
- Leon worked very hard for 4 days to complete the project
- There were some issues, so now he needs to debug the code

Modules Visualized

Consider the following situation:

- Since he wrote the whole code in one file, he is now stressed out with the fact that the code file has become too big
- Making changes in that will mess up everything

Modules Visualized

Leon is smart!

☺

- To avoid this kind of scenarios, Leon decided to divide the project into different parts based on some features or components
- Thus, he finished debugging without messing up the rest of the codes by performing it on one part of the code at a time

Why do we need Modules?

Three major points:

Easier Ways of Code Reusability

System Namespace Partitioning

Implementing Shared Services/Data

Modules

- The module is a simple **Python file** which can contain functions, variables, classes, etc.
- Modules are processed with two new statements and one important built-in function, which are as follows:

- **import**: Lets a client obtain a module
- **from**: Permits clients to fetch names from a module
- **reload**: Gives a way to reload a code of module without stopping Python

Importing a Module

```
import <file-name1, file-name2….file-namen
```

Example

```
def intellipaat():
print “Hello intellipaat”
```

- Save this file using the **py** extension
- We have saved the above script under the name **hello.py**

Modules

Remember the hello.py? Let’s use it now:

A New File

```
import hello
hello.intellipaat()
```

Output

```
Hello intellipaat
```

From … import is used to import an attribute from a module

Syntax

```
from  module-name import
atr1,atr2,…atrn
```

To import the whole module, use the following syntax:

Syntax

```
from module-name import *
```

Module Search Path

Expanding Python Search Path to Other Sources

- Set the environment variable PYTHONPATH to a colon-separated list of directories to search for imported modules
- In your program, use **sys.path.append('/path/to/search')** to add the names of directories you want Python to search for imported modules
- **sys.path** is just the list of directories Python searches every time it is asked to import a module, and you can alter it as needed
- Any directories you put in the environment variable PYTHONPATH will be inserted into **sys.path** when Python starts up

Standard Libraries in **Python**

Standard Libraries in Python

There are 10 important standard library modules:

Modules

- time
- sys
- os
- math
- random
- pickle
- urllib
- re
- cgi
- socket

Standard Libraries in Python

Python time module:

- Python has a module named **time** to handle time-related tasks
- To use functions defined in the module, you need to import the module

Import

```
import time
```

Python time.time()

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

Python time.sleep()

```
import time
print("This is printed immediately.")
time.sleep(2.4)
print("This is printed after 2.4
seconds.")
```

Standard Libraries in Python

Python sys module:

- Python has a module named **sys** to handle module-related tasks
- To use functions defined in the module, you need to import the module

Import

```
import sys
```

Python sys

```
>>> import sys
>>> for i in (sys.stdin, sys.stdout,
sys.stderr):
...     print(i)
```

Standard Data Streams

stdin	Standard Input Stream
stdout	Standard Output Stream
stderr	Standard Error Stream

Standard Libraries in Python

Python os module:

- Python has a module named **os** to handle function-related tasks
- This module provides a portable way of using OS-dependent functionalities

Import

```
import os
```

Python os

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

Module Functions

os.name	Returns the name of the dependant module
os.error	Alias for the built-in OSError exception
os.close	Used to close the file descriptor

Standard Libraries in Python

Python math module:

- Python has a module named **math** to handle math-related operations
- This module contains a plethora of mathematical functions

Import

```
import math
```

Module Functions

ceil()	Returns the smallest int value greater than the number
floor()	Returns the greatest int value smaller than the number
factorial()	Returns the factorial of the input integer number

Standard Libraries in Python

Python random module:

- Python has a module named **random** to handle number generation-related operations
- This module contains a plethora of options when generating random data required for operations

Import

```
import random
```

Module Functions

choice()	Used to generate 1 random number from a container
random()	Returns a float random number less than 1 or **>=** 0
shuffle()	Shuffles the entire list to randomly arrange the data

Standard Libraries in Python

Python pickle module:

- Python pickle module is used for serializing and de-serializing a Python object structure
- Any object in Python can be pickled so that it can be saved on the disk

Import

```
import pickle
```

Pickle keeps track of all the objects that it has already serialized

Pickle stores data once and ensures references point to master copy

If the class definition is importable, Pickle can easily save and restore class instances

Standard Libraries in Python

Python urllib module:

- The **urllib** module is the URL-handling module for Python
- It is used to fetch URLs

urllib Modules

urllib.request	Helps define functions and classes to open URLs (usually HTTP)
urllib.parse	Helps define functions to manipulate URLs and their component parts
urllib.error	If there is an error in fetching URLs, this helps in raising exceptions

Standard Libraries in Python

Python re module:

- The **re** module is the regular expression-handling module for Python
- Regular expression is a special text string used for describing a search pattern

re Methods

re.match()	Used to match the RE pattern to a string (can use flags)
re.search()	Used to find a pattern in a search text string
re.findall()	Used to iterate over a couple of lines in a file or a document

Standard Libraries in Python

Python cgi module:

- The **cgi** module is a standard used in programs when they need to interact with a client through a web server
- CGI is the standard for programs to interface with HTTP servers

Example

```
#!/usr/bin/python

print "Content-type:text/html\r\n\r\n"
print '<html>'
print '<head>'
print '<title>Hello Word - First CGI Program</title>'
print '</head>'
print '<body>'
print '<h2>Hello Word! This is my first CGI program</h2>'
print '</body>'
print '</html>'
```

Standard Libraries in Python

Python socket module:

- **Socket programming** is a way of connecting two nodes on a network to communicate with each other
- Socket programming is started by importing the socket library and making a simple socket

Import

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Connecting to the Server

```
$ ping www.google.com
```

→

Finding the IP Address

```
import socket

ip = socket.gethostbyname('www.google.com')
print ip
```

Command-line Arguments in **Python**

Command-line Arguments

What is sys.argv?

- **sys.argv** is the list of command-line arguments passed to the Python program
- **argv** represents all the items that come along via the command-line input
- Remember that Python **arrays** index from 0 and not from 1!

Reading an argument from cmd line

```
import sys
for x in sys.argv:
     print "Argument: ", x
```

len(sys.argv): Checks the number of arguments entered

Example

```
import sys
program_name = sys.argv[0]
arguments = sys.argv[1:]
count = len(arguments)
```

sys.argv[0]: Denotes the name of the program always!

sys.argv[1]: Denotes the 1st argument passed to the program

Exceptions in **Python**

Python Exceptions

What is an Exception?

- An exception is an error that happens during the execution of a program
- When an error occurs, Python generates an exception which ensures that the program does not crash

Why use Exceptions?

- Exceptions are convenient when handling errors and special conditions
- If there is a scenario which requires the code to be checked, you can make use of exception handling

Raising an Exception

- The user can raise an exception in the program by making use of simple syntax
- Once the exception is raised, the compiler expects the user to handle it and only then it will continue the execution

Common Exceptions

IO Error	If the requested file cannot be opened
Import Error	If Python cannot find the respective module
EOF Error	If a built-in function hits EOF without data

Python Exceptions

Setting up your first Exception block:

- **try:**
  Statements here
- **except:**
  exception handling

Example

```
try:
    a=1/0
    print(a)

except ZeroDivisionError:
    print ("You can't divide by zero!!!")
```

Output

Python Exceptions

Try – Except – Else Clause

- The **else** clause in a **try** - **except** statement must follow all the except clauses and is useful when the try clause does not raise the exception

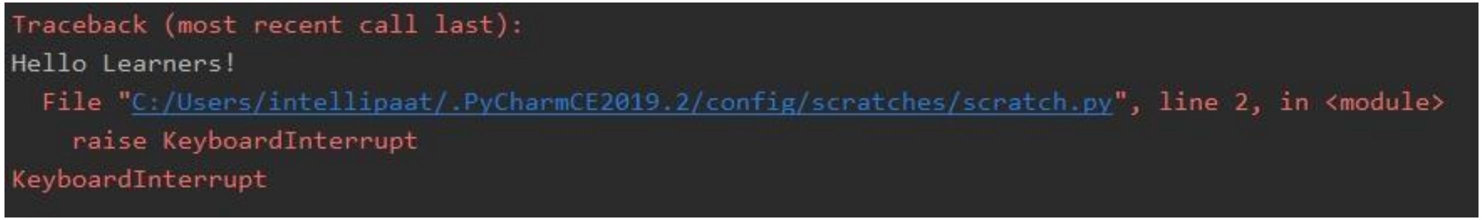
```
try:
   data = something_that_can_go_wrong

except IOError:
   handle_exception_error

else:
   different_exception_handling
```

Python Exceptions

Try – Finally Clause

- The finally clause is **optional**
- It is intended to define the clean-up actions that must be executed **no matter what!**

```
try:
    raise KeyboardInterrupt
finally:
    print ('Hello Learners!')
```



```
Traceback (most recent call last):
Hello Learners!
  File "C:/Users/intellipaat/.PyCharmCE2019.2/config/scratches/scratch.py", line 2, in <module>
    raise KeyboardInterrupt
KeyboardInterrupt
```

Installing Packages in **Python**

Installing Packages in Python

Using **pip**, the packages can be installed in Python easily

- Python language does **not** come pre-installed with the fancy features you might want for your usage
- When you need a functionality, you can look toward **Python packages**
- A package structures Python modules, which contain **pre-written code** that other developers have created for you
- Modules are **handy** when you are looking for a **specific** functionality

```
pip install scrapy
```

Installing Packages in Python

Installing from other indexes:

- Installing from an alternate index:

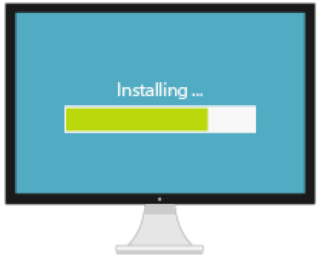
```
pip install --index-url http://my.package.repo/simple/ YourProject
```

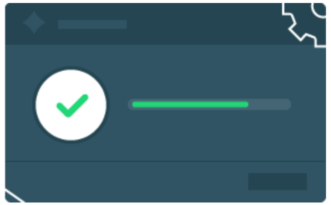
- Installing from a local src tree:

```
pip install -e <path>
```

- Installing Prereleases:

```
pip install --pre YourProject
```





Quiz!

Question 1

Which of the following is an OOP concept?

A	Data Type
B	Identifier
C	Encapsulation
D	End of File (EOF)

Answer 1

Which of the following is an OOP concept?

A	Data Type
B	Identifier
C	**Encapsulation** ✓
D	End of File (EOF)

Question 2

Functions can be anonymously invoked using the Lambda function.

A True

B False

Answer 2

Functions can be anonymously invoked using the Lambda function.

A	True ✓
B	False

Question 3

What is the use of the **cgi** module in Python?

A	Connects a Client to a Web Server
B	A Computational Graphics Module
C	Graph Construction and Analysis
D	Gaming application development

Answer 3

What is the use of the **cgi** module in Python?

A	Connects a Client to a Web Server ✓
B	A Computational Graphics Module
C	Graph Construction and Analysis
D	Gaming application development

+91-9030485102

rganesh0203@gmail.com

https://topmate.io/rganesh_0203