Day – 5 & 6 & 7

Transforming Engineers to Technocrats

Variable Scopes

- Not all variables are accessible from all parts of our program.
- The part of the program where the variable is accessible is called its “scope” and is determined by where the variable is declared.
- Python has three different variable scopes:
  - Local scope
  - Global scope
  - Enclosing scope

Local Scope

```python
def myfunc():
    x = 42        # local scope x
    print(x)

myfunc()          # prints 42
```

```python
def myfunc():
    x = 42        # local scope x

myfunc()
print(x)          # Triggers NameError: x does not exist
```

Global Scope

```python
x = 42          # global scope x

def myfunc():
    print(x)    # x is 42 inside def

myfunc()
print(x)        # x is 42 outside def
```

```python
x = 42          # global scope x
def myfunc():
    x = 0
    print(x)    # local x is 0

myfunc()
print(x)        # global x is still 42
```

```python
verbose = True

def op1():
    if verbose:
        print('Running operation 1')
```

```python
x = 42          # global scope x
def myfunc():
    global x    # declare x global
    x = 0
    print(x)    # global x is now 0

myfunc()
print(x)        # x is 0
```

Enclosing Scope

```python
# enclosing function
def f1():
    x = 42
    # nested function
    def f2():
        x = 0
        print(x)    # x is 0
    f2()
    print(x)        # x is still 42

f1()
```

```python
# enclosing function
def f1():
    x = 42
    # nested function
    def f2():
        nonlocal x
        x = 0
        print(x)    # x is now 0
    f2()
    print(x)        # x remains 0

f1()
```

Generator Functions

- Generators are iterable functions.
- Generate values once at a time from a given sequence, instead of giving entire sequence.
- yield keyword is used to return values from generator functions.
- We need to call next() function in order to get values one by one from generator functions.
- After returning all values one by one next() function raises StopIteration error, indicating all values are taken out.
- Example:

```python
def gnrtrFunc():
  for i in range(1,11):
    yield i**2

values=gnrtrFunc()
print(type(values))
print(values)

print(next(values))

for i in values:
  print(i)
```

```
<class 'generator'>
<generator object gnrtrFunc at 0x000001A93D9FD468>
1
4
9
16
25
36
49
64
81
100
>>>
```

Decorator Functions

- Sometimes you want to modify an existing function without changing its source code. A common example is adding extra processing (e.g. logging, timing, etc.) to the function.
- That's where decorators come in.
- A decorator is a function that accepts a function as input and returns a new function as output, allowing you to extend the behavior of the function without explicitly modifying it.
- In Python, a function can be:
  - Assigned to a variable
  - Passed as argument
  - Defined inside another function
  - Returned from a function

Function Decorators

- Syntax

```
@funcDcrtr
def dcrtrFunc():
        print(“func”)
```

- Is equivalent to

```
def dcrtrFunc():
    print(“func”)
dcrtrFunc = funcDcrtr(dcrtrFunc)
```

```python
def decorate_it(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper

def hello():
    print("Hello world")

hello = decorate_it(hello)

hello()
# Prints Before function call
# Prints Hello world
# Prints After function call
```

9

Modules & Packages

Python Functions

Creating Python Module

```python
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```

```python
>>> import fibo
```

```python
>>> fibo.fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'
```

Python Module Import

- Module is a collection of functions, classes and variables packaged in a file.
- Enables code reusability.
- Can be brought in using import, from, as keywords.
- **Syntax1**

```
import <ModuleName>
#this will import the source file ModuleName in another file.
```

- **Syntax2**

```
from <ModuleName> import name1[,name2[,nameN]]
#this will import specific attributes from a ModuleName into current file.
```

- **Syntax3**

```
from <ModuleName> import *
#this will import all attributes from a ModuleName into current file.
```

- **Syntax4**

```
import <ModuleName> as <NewName>
#this will import ModuleName renamed as NewName
```

Setting Module Search Path

```
import sys
sys.path.append('/home/test/')

import calculation
print(calculation.add(1,2))
```

Installing Python Modules

- `python –m pip install <module>`
- Or
- `pip install <module>`

- For multiple python versions, specify python version
- To install PIP (Python Package Installer)
  - Download `get-pip.py`
  - Run
    - `python get-pip.py`

Common Modules

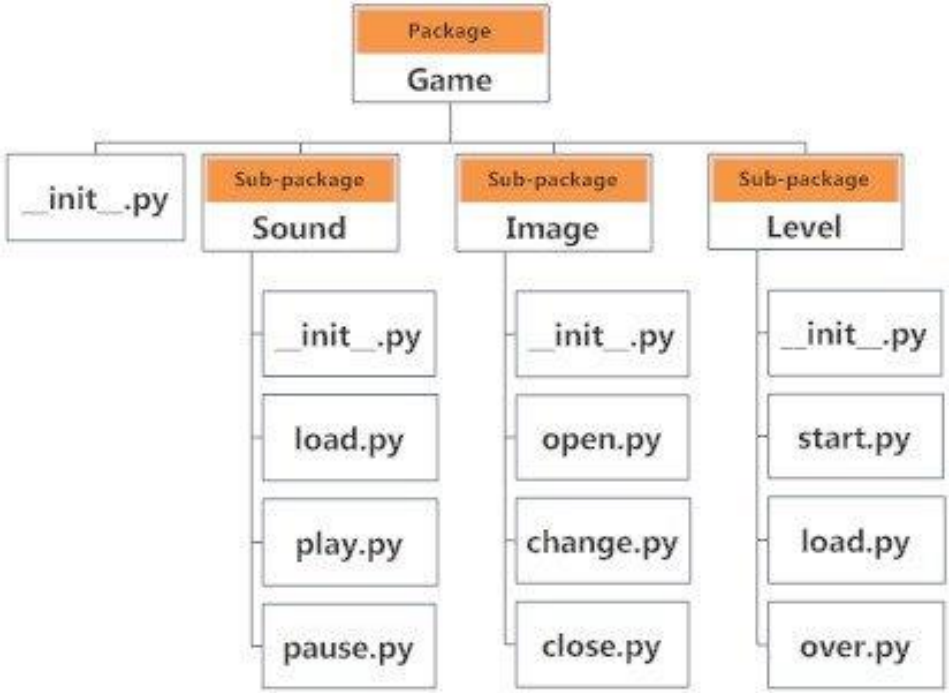
- os
- sys
- math
- json
- csv
- numpy
- pandas
- sklearn
- tkinter

- Using dir() function

```
>>> import fibo, sys
>>> dir(fibo)
['__name__', 'fib', 'fib2']
>>> dir(sys)
['__breakpointhook__', '__displayhook__', '__doc__', '__excepthook__',
 '__interactivehook__', '__loader__', '__name__', '__package__', '__spec__',
 '__stderr__', '__stdin__', '__stdout__', '__unraisablehook__',
 '_clear_type_cache', '_current_frames', '_debugmallocstats', '_framework',
 '_getframe', '_git', '_home', '_xoptions', 'abiflags', 'addaudithook',
 'api_version', 'argv', 'audit', 'base_exec_prefix', 'base_prefix',
 'breakpointhook', 'builtin_module_names', 'byteorder', 'call_tracing',
 'callstats', 'copyright', 'displayhook', 'dont_write_bytecode', 'exc_info',
 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info',
 'float_repr_style', 'get_asyncgen_hooks', 'get_coroutine_origin_tracking_depth',
 'getallocatedblocks', 'getdefaultencoding', 'getdlopenflags',
 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getprofile',
 'getrecursionlimit', 'getrefcount', 'getsizeof', 'getswitchinterval',
 'gettrace', 'hash_info', 'hexversion', 'implementation', 'int_info',
 'intern', 'is_finalizing', 'last_traceback', 'last_type', 'last_value',
 'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks',
 'path_importer_cache', 'platform', 'prefix', 'ps1', 'ps2', 'pycache_prefix',
 'set_asyncgen_hooks', 'set_coroutine_origin_tracking_depth', 'setdlopenflags',
 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace', 'stderr',
 'stdin', 'stdout', 'thread_info', 'unraisablehook', 'version', 'version_info',
 'warnoptions']
```

Python Packages

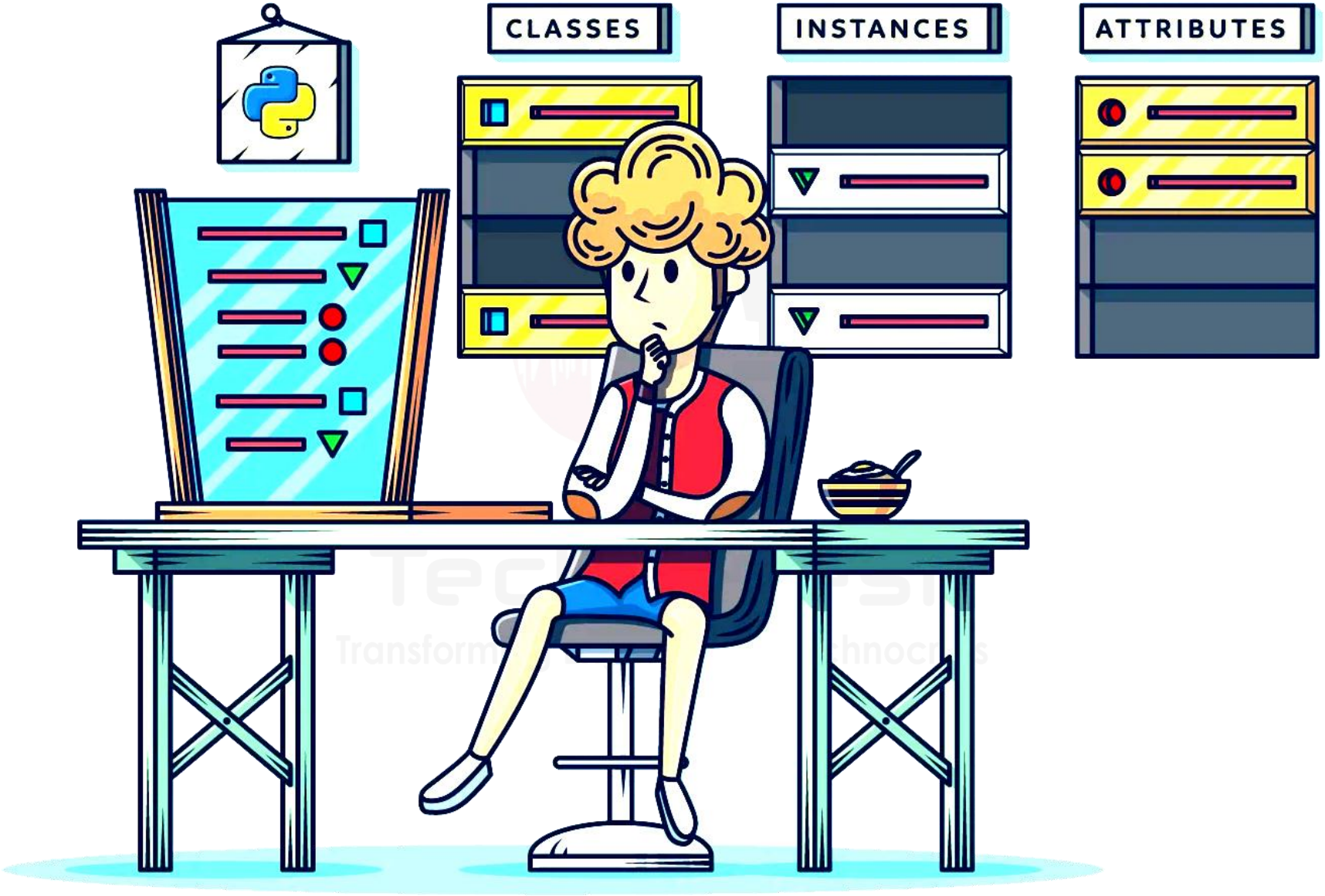
- Packages are a way of structuring Python's module namespace by using "dotted module names".
- Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files.
- As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.
- Similarly, as a directory can contain subdirectories and files, a Python package can have sub-packages and modules.
- A directory must contain a file named __init__.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.



16

Object Oriented Programming

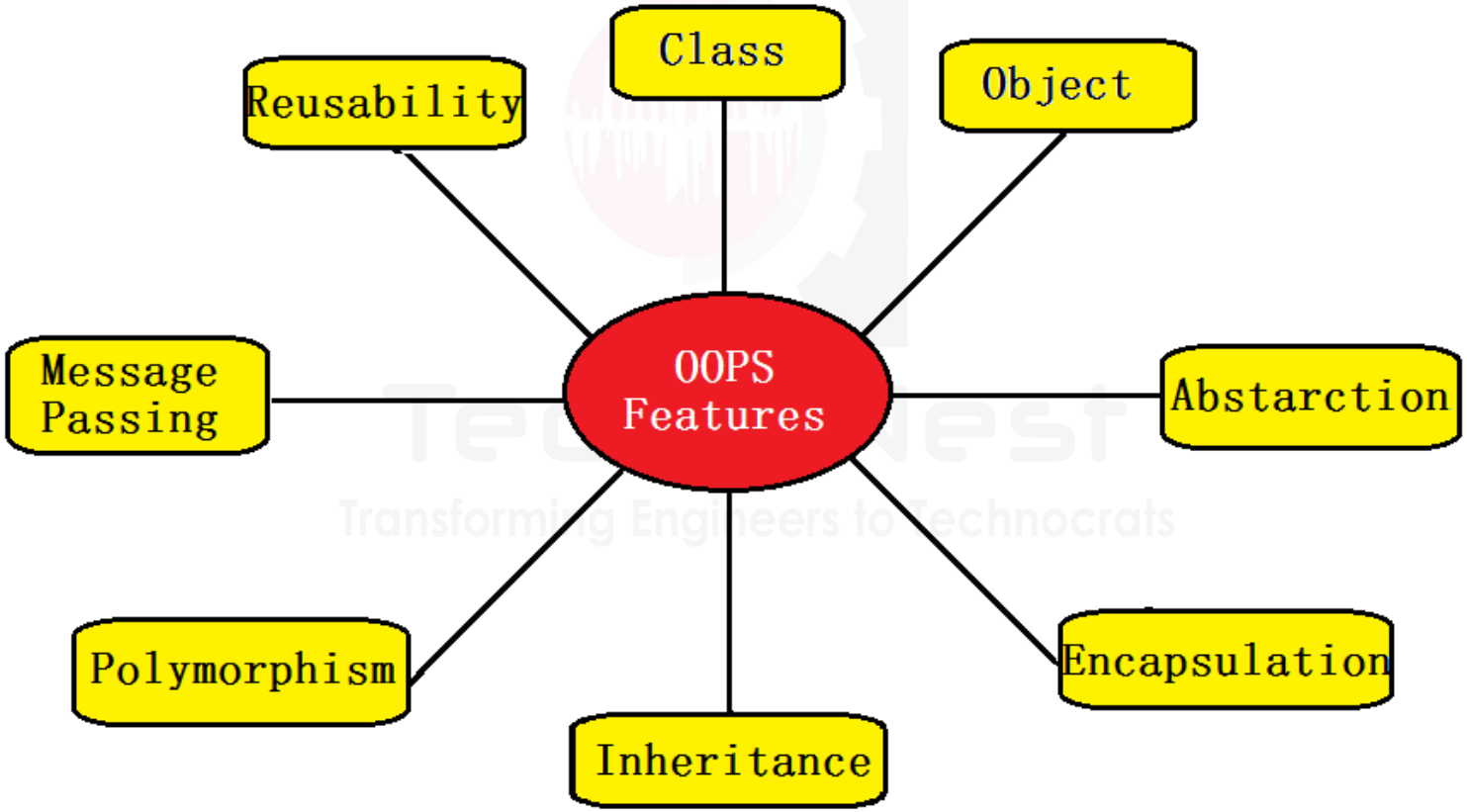
OOP


CLASSES
INSTANCES
ATTRIBUTES

OOP

- **Object-oriented Programming**, or OOP for short, is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- Object-oriented programming is an approach for modeling concrete, real-world things like cars as well as relations between things like companies and employees, students and teachers, etc.
- OOP models real-world entities as software objects, which have some data associated with them and can perform certain functions.
- Another common programming paradigm is ***procedural programming*** which structures a program like a recipe in that it provides a set of steps, in the form of functions and code blocks, which flow sequentially in order to complete a task

OOP Features



OOP – Terminologies

- **Class:** It is a kind of template, which comprises of multiple attributes and methods (called as its members also)

```
class Parrot:
        pass
```

- **Object:** Its an instance (handle) of a class. With this, we can access all the features of any class.

```
obj = Parrot()
```

- **Methods:** Methods are functions defined inside the body of a class. They are used to define the behaviors of an object

```
def function(self):
        pass
```

- **Inheritance:** Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class)

OOP – Terminologies contd.

- Encapsulation: Using OOP in Python, we can restrict access to methods and variables. This prevent data from direct modification which is called encapsulation. In Python, we denote private attribute using underscore as prefix i.e single “ _ “ or double “ __“.
- **Polymorphism:** Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.

Defining a Class in Python

```
class Parrot():
        print('Class created')
        a = 20
        b = 30

obj = Parrot()
print(obj)
print(obj.a)
print(obj.b)
```

Defining a Method in Python

```
class Parrot():
        print('Class created')
        a = 20
        b = 30
        def function(self):
                print("Great")

obj = Parrot()
obj.function()
```

Attributes in Classes

- Instance Attributes
  - These are the ones which have separate values for separate objects. (Initialized with default values in all, if given)
  - self.name
- Class Attributes
  - These are common for all class members, objects. If one tries to change them, they get changed for all.
  - ClassName.name

Methods in Classes

- User Defined Methods
  - The methods which we define by our own for custom purposes
  - E.g.

```
def function (self):
    pass
```

- Dunder Functions (Magic Functions)
  - The pre-built name of methods, which actually covered under double underscores as leading and trailing characters

```
def __init__(self):
    print('Hi')
```

```python
class Employee: #class created
  id = 0     #property (variable)
  name = "" #property (variable)
  def createEmp(self,id,name): #behavior (method)
    self.id = id
    self.name = name

  def showInfo(self):  #behavior (method)
    print("Id:",self.id)
    print("Name:",self.name)


emp1 = Employee() #object1
emp2 = Employee() #object2

emp1.createEmp(1,"ABC")  #access behavior
emp2.createEmp(2,"XYZ")  #access behavior

emp1.showInfo()  #access behavior
emp2.showInfo()  #access behavior

emp1.name = "PQR"  #access property

emp1.showInfo()  #access behavior
emp2.showInfo()  #access behavior
```

```
Python 3.6.5 (v3.6.5
4)] on win32
Type "copyright", "
>>>
================== R
Id: 1
Name: ABC
Id: 2
Name: XYZ
Id: 1
Name: PQR
Id: 2
Name: XYZ
>>>
```

Class Attributes

File Edit Format Run Options Window Help

```
class Student:
  count = 0  #static variable
  def __init__(self):
    Student.count += 1  #must be accessed using class name

s1 = Student()
s2 = Student()
s3 = Student()

print("Number of students:",Student.count)
```

File Edit Shell Debug Options Win

```
Python 3.6.5 (v3.6.5:f59
4)] on win32
Type "copyright", "cred
>>>
================== RESTA
Number of students: 3
>>>
```

Python Dunder Functions

- Constructors & Destructors: `__init__`, `__del__`
- Iteration & Length: `__getitem__`, `__len__`
- Method Invocation: `__call__`
- Strings Representation: `__str__`, `__repr__`
- Operator Overloading: `__eq__`, `__lt__`, `__add__`, `__sub__`, `__mul__`, …

Constructors

- It is a special type of method which is used to initialize the instances of the class.
- Constructor is executed with creation of the objects.
- It also verify that there are enough resources for the object to perform any start-up task.
- To create constructor __init__ method is used. We can pass any number of arguments to create and initialize objects to constructor.

File Edit Format Run Options Window Help

```
class Employee: #class created
  id = 0     #property (variable)
  name = "" #property (variable)

  def __init__(self,id,name):  #constructor
    self.id = id
    self.name = name

  def showInfo(self):  #behavior (method)
    print("Id:",self.id)
    print("Name:",self.name)


emp1 = Employee(1,"ABC") #object1
emp2 = Employee(2,"XYZ") #object2

emp1.showInfo()  #access behavior
emp2.showInfo()  #access behavior

emp1.name = "PQR"  #access property

emp1.showInfo()  #access behavior
emp2.showInfo()  #access behavior
```

File Edit Shell Debug Optio

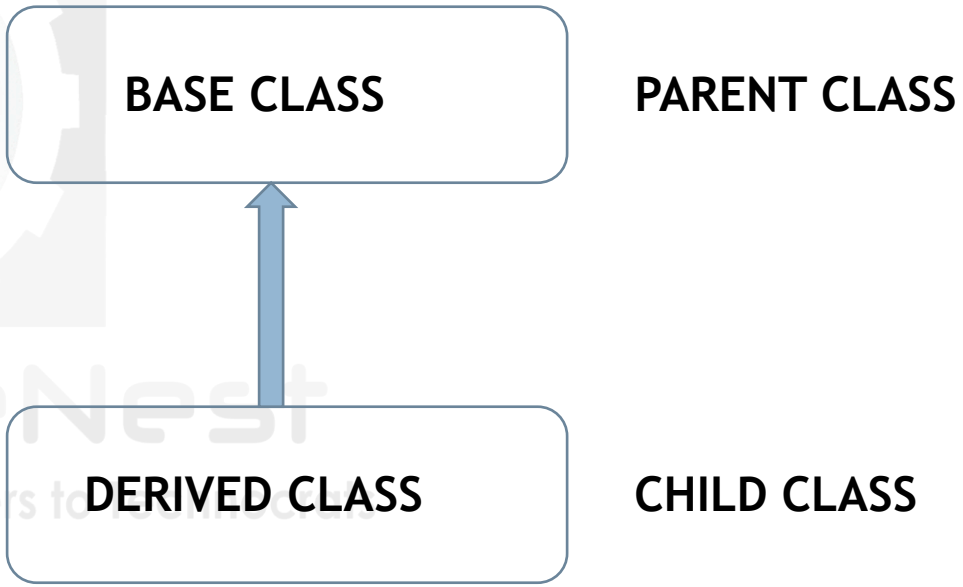
```
Python 3.6.5 (v3.6.
4)] on win32
Type "copyright", "
>>>
==================
Id: 1
Name: ABC
Id: 2
Name: XYZ
Id: 1
Name: PQR
Id: 2
Name: XYZ
>>>
```

Inheritance

- Provide code reusability because we can use existing class to create a new class.
- The child class acquires properties and can access all data members and functions in the parent
- Syntax

```
class derived-class(base-class):
    #block of class
```

- Types:
  - Single Level
  - Multilevel
  - Multiple
  - Hybrid



File Edit Format Run Options Window

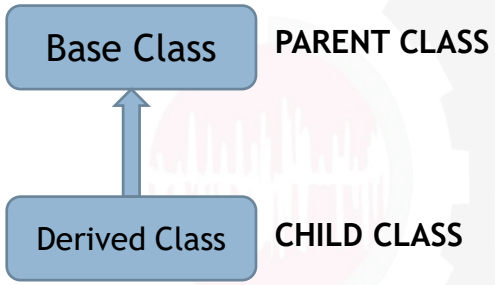
```python
class Animal:
  no_of_legs = 4
  def speak(self):
    print("Animal class")

class Dog(Animal):
  def bark(self):
    print("Dog Barking")

d = Dog()
print(d.no_of_legs)
d.speak()
d.bark()

a= Animal()
print(a.no_of_legs)
a.speak()
a.bark()
```

File Edit Shell Debug Options Window Help

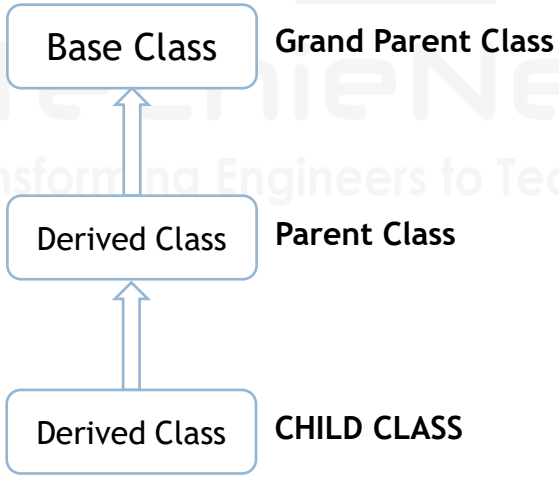
```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.
4)] on win32
Type "copyright", "credits" or "license()" for more information
>>>
================== RESTART:                                    ===
4
Animal class
Dog Barking
4
Animal class
Traceback (most recent call last):
  File "C:\Users\Himanshu\Desktop\vv.py", line 18, in <module>
    a.bark()
AttributeError: 'Animal' object has no attribute 'bark'
>>>
```

Types of Inheritance

- Single Level



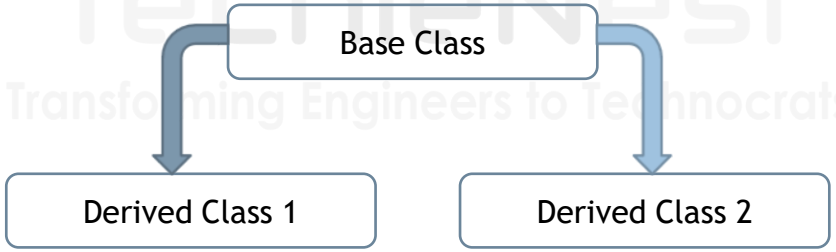
- Multilevel



Types of Inheritance

- Multiple

- Hierarchical



Polymorphism

- Polymorphism can be achieved using method overriding.
- Parent class method is defined in the child class with specific implementation.

File Edit Format Run Options Window Help

```
class Bank:
  def getroi(self):
    return 4.5

class Kotak(Bank):
  def getroi(self):
    return 6.5

class ICICI(Bank):
  def getroi(self):
    return 6.0

b1 = Bank()
b2 = Kotak()
b3 = ICICI()

print("ROI for Bank:",b1.getroi())
print("ROI for Kotak:",b2.getroi())
print("ROI for ICICI:",b3.getroi())
```

File Edit Shell Debug Options Win

```
Python 3.6.5 (v3.6.5:f5
4)] on win32
Type "copyright", "cred
>>>
================== REST
ROI for Bank: 4.5
ROI for Kotak: 6.5
ROI for ICICI: 6.0
>>>
```

Abstraction (Data Hiding)

- This can be done by '__' double under scores
- Although we can access our elements by object, obj._ClassName__data

```
File  Edit  Format  Run  Options  Window  Help
class Employee:
  __count = 0
  def __init__(self):
    Employee.__count += 1
  def display(self):
    print("Count:",Employee.__count)

emp1 = Employee()
emp2 = Employee()

emp1.display()
print(emp1.__count)
```

```
File  Edit  Shell  Debug  Options  Window  Help
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.
4)] on win32
Type "copyright", "credits" or "license()" for more information
>>>
==================                                        ===
Count: 2
Traceback (most recent call last):
  File "C:\Users\Himanshu\Desktop\vv.py", line 12, in <module>
    print(emp1.__count)
AttributeError: 'Employee' object has no attribute '__count'
>>> |
```

36

Errors & Exception Handling

Python Functions

Python Errors & Exceptions

- An exception can be defined as an abnormal condition in a program resulting in the flow of the program.
- It causes the program to halt the execution.
- So exception handling is a way to deal with this problem. So that other part of the code can be executed without any disruption.
- Common Exceptions
  - ZeroDivisionError – Occurs when a number is divided by 0.
  - NameError – Occurs when a name is not found (local or global).
  - IndentationError – Occurs when incorrect indentation is given.
  - IOError – Occurs when Input Output operation fails.
  - EOFError – Occurs when end of file is reached, and yet operations are being performed.

Python Exception Handling contd.

- Problem without handling exception
- Here, we have entered b=0.
- It raises ZeroDivisionError at c = a/b.
- It causes program to entered in halt condition.
- So in order to handle this situation, We have exception handling

File Edit Format Run Options Window Help

```
a = int(input("Enter a: "))
b = int(input("Enter b: "))

c = a/b

print("a/b =",c)

print("Hello, This is the other part of the code!!!")
```

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v
4)] on win32
Type "copyright", "credits" or "license()" for more informatic
>>>
==================                                             ==
Enter a: 25
Enter b: 0
Traceback (most recent call last):
  File "C:\Users\Himanshu\Desktop\vv.py", line 4, in <module>
    c = a/b
ZeroDivisionError: division by zero
>>>
```

Python Exception Handling contd.

- We have following keywords to handle exceptions, so that we can deal with exceptions in several ways
  - try
  - except
  - else
  - finally
  - raise
- The try and except keywords

```
try:
      Test This Code For An Exception
except:
      Run This Code If An Exception Occurs
```

- We can have multiple except blocks associated with one try block.

File Edit Format Run Options Window Help

```python
a = int(input("Enter a: "))
b = int(input("Enter b: "))

try:
  c = a/b
except ZeroDivisionError:
  print("Denominator can't be zero, Enter again!!!")
  b = int(input("Enter b: "))
  c = a/b

print("a/b =",c)

print("Hello, This is the other part of the code!!!")
```

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
4)] on win32
Type "copyright", "credits" or "license()" fo
>>>
==================
Enter a: 25
Enter b: 0
Denominator can't be zero, Enter again!!!
Enter b: 12
a/b = 2.0833333333333335
Hello, This is the other part of the code!!!
>>>
```

Python Exception Handling contd.

- The else keyword

```
try:
        #test code
except Exception:
        #handle exception, if any
else:
        #do this if no exception raised
```

```python
a = int(input("Enter a: "))
b = int(input("Enter b: "))

try:
  c = a/b
except ZeroDivisionError:
  print("Denominator can't be zero, Enter again!!!")
  b = int(input("Enter b: "))
  c = a/b
else:
  print("Well Done, No exception!!!")

print("a/b =",c)

print("Hello, This is the other part of the code!!!")
```

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018,
4)] on win32
Type "copyright", "credits" or "license()" for
>>>
==================
Enter a: 25
Enter b: 12
Well Done, No exception!!!
a/b = 2.0833333333333335
Hello, This is the other part of the code!!!
>>>
```

Python Exception Handling contd.

- The **finally** keyword

```
try:
        #test code
except Exception:
        #handle exception, if any
finally:
        #always run this code
```

File Edit Format Run Options Window Help

```
a = int(input("Enter a: "))
b = int(input("Enter b: "))

try:
  c = a/b
except ZeroDivisionError:
  print("Denominator can't be zero, Enter again!!!")
  b = int(input("Enter b: "))
  c = a/b
finally:
  print("I will always run!!!")

print("a/b =",c)

print("Hello, This is the other part of the code!!!")
```

File Edit Shell Debug Options Window Help

```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 
4)] on win32
Type "copyright", "credits" or "license()" for
>>>
==================
Enter a: 25
Enter b: 0
Denominator can't be zero, Enter again!!!
Enter b: 8
I will always run!!!
a/b = 3.125
Hello, This is the other part of the code!!!
>>>
================== RESTART: C:\Users\Himanshu\
Enter a: 25
Enter b: 8
I will always run!!!
a/b = 3.125
Hello, This is the other part of the code!!!
>>>
```

Python Exception Handling contd.

- The raise keyword :- Exception can be raised using following syntax-
- `raise Exception_class, <value>`

File Edit Format Run Options Window Help

```
try:
  age = int(input("Enter the age: "))
  if age<18:
    raise ValueError
  else:
    print("The age is valid!!!")
except ValueError:
  print("The age is not valid!!!")
```

File Edit Shell Debug Options Window

```
Python 3.6.5 (v3.6.5:f59c09
4)] on win32
Type "copyright", "credits"
>>>
================== RESTART:
Enter the age: 15
The age is not valid!!!
>>>
================== RESTART:
Enter the age: 18
The age is valid!!!
>>>
```

44

File Handling

Python Functions

Simple File Handling

- Python can handle files with following commands

```
with open(‘myfile’, ‘r’) as f:
     r = f.read()
     f.close()
with open(‘myfile’, ‘w+’) as f:
     f.write(‘Mydata String’)
     f.close()
with open(‘myfile’, ‘a+’) as f:
     f.write(‘More data strings’)
     f.close()
```

File Modes

MODE	DESCRIPTION
r	Read only mode, open text file, pointer location beginning, file must exist.
rb	Read only mode, open binary file, pointer location beginning, file must exist.
r+	Read & Write mode, open text file, pointer location beginning, file must exist.
rb+	Read & Write mode, open binary file, pointer location beginning, file must exist.
w	Write only mode, open text file, pointer location beginning, file created if not exist.
wb	Write only mode, open binary file, pointer location beginning, file created if not exist.
w+	Write and read mode, open text file, pointer location beginning, file created if not exist.
wb+	Write and read mode, open binary file, pointer location beginning, file created if not exist.
a	Append only mode, open text file, pointer location end of file, file created if not exist.
ab	Append only mode, open binary file, pointer location end of file, file created if not exist.
a+	Append and read mode, open text file, pointer location end of file, file created if not exist.
ab+	Append and read mode, open binary file, pointer location end of file, file created if not exist.
x	Create a new file.