Encapsulation

Encapsulation is the process of using private variables within classes to prevent unintentional or potentially malicious modification of data. By containing and protecting variables within a class, it allows the class and the objects that it creates to function as independent, self-contained, parts functioning within the machine of the program itself.

Through encapsulation variables and certain methods can only be interacted with through the interfaces designated by the class itself.

Python has different levels of restriction that control how data can be accessed and from where. Variables and methods can be public or private. Those designations are made by the number of underscores before the variable or method.

Public
Public variables and methods can be freely modified and run from anywhere, either inside or outside of the class.

Private
The private designation only allows a variable or method to be accessed from within its own class or object. You cannot modify the value of a private variable from outside of a class. Private variables and methods are preceded by two underscores

```
class Car(object):
        def __init__(self, make = 'Ford', model = 'Explorer', year = '2019', color = 'blue'):
        self.__make = make
        self.__model = model
        self.__year = year
        self.__color = color

    def getModel(self):
         return(self.__model)


# start the program
mycar = Car()

# The statement below will not work because the attribute is private
#print(mycar.__model)

# You must use the class getter method to access the attribute
print(mycar.getModel())
```

Polymorphism

Polymorphism allows subclasses to have methods with the same names as methods in their super-classes. It gives the ability for a program to call the correct method depending on the type of object that is used to call it.

```
''' This program demonstrates the concepts of Polymorphism '''

class India():
    def capital(self):
        print("The capital of India is New Delhi.")

    def language(self):
        print("Hindi is the primary language of India.")

class USA():
    def capital(self):
        print("The capital of USA is Washington, D.C.")

    def language(self):
        print("English is the primary language of USA.")

def func(obj):
    obj.capital()
    obj.language()

# Create a list of countries
lst = []
lst.append(India())
lst.append(USA())

# print each country's information
for country in range (0, len(lst)):
      func(lst[country])
      print("\n")
```

```
The capital of India is New Delhi.
Hindi the primary language of India.

The capital of USA is Washington, D.C.
English is the primary language of USA.
```

Abstract Classes

Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation. Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods. Subclasses of an abstract class in Python are not required to implement abstract methods of the parent class.

In fact, Python on its own does not provide abstract classes. Yet, Python comes with a module which provides the infrastructure for defining Abstract Base Classes (ABCs). This module is called for obvious reasons - ABC.

The following Python code uses the ABC module and defines an abstract base class:

```
from abc import ABC, abstractmethod

class Polygon(ABC):

    # abstract method
    def noofsides(self):
        pass

class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")
```

```
# Start of program
lst = []  # create a lits of shapes
lst.append(Triangle())
lst.append(Quadrilateral())
lst.append(Pentagon())
lst.append(Hexagon())

# print each shape's number of sides
for i in range(0, len(lst)):
    lst[i].noofsides()
```

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

```
from abc import ABC, abstractmethod

class Vehicle(ABC):
    '''This class inherits from (or subclasses) ABC'''
    @abstractmethod
    def number_of_wheels(self):
        '''This method is abstract, so the class cannot be
           instantiated. This method will be overridden in
           subclasses of Vehicle'''
        pass

class Car(Vehicle):
    '''This class inherits from the abstract base class Vehicle'''
    def number_of_wheels(self):
        '''Override the abstract method in the base class'''
        return 4

# create a car: successful
honda = Car()

# print the number of wheels: successful
print(honda.number_of_wheels())

# Try to create a Vehicle: failed.
#v = Vehicle()
```

'''
See error message below:

TypeError: Can't instantiate abstract class Vehicle
with abstract methods number_of_wheels
'''

Python *isinstance()* function

''' This program demonstrates Python isinstance() function '''

```
class Employee:

  def __init__(self, name, salary):
    self.name = name
    self.salary = salary

emp = Employee("Emma", 11000)

print("Checking emp object is an instance of Employee")
if isinstance(emp, Employee) :
  print("Yes! given object is an instance of class Employee\n")
else:
  print("No! given object is not an instance of class Employee


firstNumber = 80
result = isinstance(firstNumber, (int, float))
print(firstNumber,'is an instance of int or float?', result)

name = 'Python'
flag = isinstance(name, str)
print(name,'is an instance of String?', flag, "\n")

sampleList = ['Eric', 'Scott', 'Kelly']
flag = isinstance(sampleList, list)
print(sampleList,'is instance of list?', flag)
```

Test-run

```
Checking emp object is an instance of Employee
```

```
Yes! a given object is an instance of class Employee
80 is an instance of int or float? True
Python is an instance of String? True
['Eric', 'Scott', 'Kelly'] is instance of list? True
```