Object Oriented Programming in Python

- Class is a model to create Objects
- Variables represent Attributes and Methods represent actions performed.
- Methods and variables are also available in Objects aka Instance Variables.
- A function written inside a class is known as method.
    - A method can be called in 2 ways:
        - classname.methodname()
        - instancename.methodname()

Creating a Class

```
class Classname(object):

        attributes

        def __init__(self):

        def method1():

        def method2():
```

Object represents the base class from where all classes in python are derived.However writing object is not compulsory.

__init__(self) is a special method used to initialize the variables.

```
class Student:
    def __init__(self):
        self.name='Vishnu'
        self.age=20
        self.marks=900
    def talk(self):
        print('Hi, I am',self.name)
        print('My age is',self.age)
        print('My marks are',self.marks)
```

Observations from the code

- Class name always begins with a capital letter.
- __init__() method is used to initialise variables.
- It has 2 underscores before and after. This indicates that method is internally defined and cannot be called explicitly.
- Self is a variable referring to the current class instance.
- Name , age and marks are instance variables.
- To refer instance variables , we can use .(dot) operator along with self.

- method talk() takes self variable as parameter.
- The methods that acts on instance(objects) of a class are known as instance methods.
- Instance methods use self as the first parameter that refers to the location of the instance in the memory.
- __init__(self) and talk(self) both are instance methods.

- To use a class , we should create an instance.
- Instance creation represents allocating memory necessary to store the actual data of variables.
- Vishnu ,20 and 900 are the data.

Syntax of creating instance:

instancename= Classname()

s1=Student()

After this the following things happen sequentially:

-Block of memory allocated on heap.

-special method __init__(self) is called internally. Also called constructor.

-Allocated memory location address of instance is returned into s1.

s1.name , s1.age, s1.marks , s1.talk() can refer to instances.

Sample Program

```
class Student:
        def __init__(self):
            self.name='Vishnu'
            self.age=20
            self.marks=900
    def talk(self):
        print('Hi, I am',self.name)
        print('My age is',self.age)
        print('My marks are',self.marks)
s1= Student()
s1.talk()
```

Self variable

- Self is a default variable that contains the memory address of the instance of the current class.
- We can refer to all instance variables and instance methods.
- The memory location of an instance created is internally passed to self.
- So self can refer to all the members of the instance.
- It is used as the first parameter in constructor and other instance method.
- If a method wants to act on instance variables it must contain ‘self’.

Constructor

- Special method to initialize the instance variables of a class.
- First parameter will be always ‘self’.
- It is called automatically when an object(instance)is created.
  - e.g. s1= Student()
- We can also pass some value to a constructor like
  - s1 = Student(‘Madhav’,123)
- Def __init__(self, n=’ ’, m=0):
  - self.name=n
  - self.marks=m
- Here formal arguments are n(none) and m(zero). Hence if we do not pass any values to constructor at the time of creating an instance, the default values of these formal arguments are stored into name and marks variables.

```
Class Student:
        def __init__(self, n=' ',m=0):
                self.name=n
                self.marks=m
        def display(self):
                print('Hi', self.name)
                print('Your marks', self.marks)
s= Student()
s.display()
print('-----------------------------------')

#constructor called with 2 arguments
s1=Student('Madhav Vyas', 123)
s1.display()
print('-----------------------------------')

# execute and see the above code
```

- Constructors are used to initialize or store beginning values into instance variables.
- It is called only once per each instance. So three times it is called if there are 3 instance created.

There are 2 types of variables:

Instance Variables

Class Variables or Static variables

Program to understand class variables or static variables

```
class Sample:
        x=10    #this is a class variable(static)
        @classmethod   #this is a decorator to mark this method as class method
        def modify(cls):  #cls must be the first parameter
                cls.x+=1   #class variable can be access using ‘cls’ instead of classname.variable
s1= Sample()
s2= Sample()
print(‘x in s1=’,s1.x)
print(‘x in s2=’,s2.x)

s1.modify()
print(‘x in s1=’,s1.x)
print(‘x in s2=’,s2.x)
```

clsvar.py

Types Of Methods

-Instance Methods-->Methods that act upon instance variables of the class.

--Accessor Methods--Simply reads data of variables, getter()

--Mutator Methods--Modify data, setter()

-Class Methods

-Static Methods

```
class Student:
        def setName(self,name):
                self.name=name
        def getName(self):
                return self.name

        def setMarks(self,marks):
                self.marks=marks
        def getMarks(self):
                return self.marks

n=int(input('How many students?'))
i=0
while(i<n):
                s= Student()
                name=input('Enter name:')
                s.setName(name)
                marks=int(input('Enter marks :'))
                s.setMarks(marks)
                print('Hi ',s.getName())
                print('your marks', s.getMarks())
                i+=1
                print('----------------------------------')
```

Class Methods

- These methods act on class level.
- Written using @classmethod decorator above them
- First parameter must be ‘cls’
- The processing needed by all instances are handled by class methods.

example

```
class Bird:
        wings=2   #class var
        @classmethod
        def fly(cls,name):
                print(‘{} flies with {} wings’.format(name,cls.wings))
Bird.fly(‘Sparrow’)
Bird.fly(‘Pigeon’)
```

Static Methods:

They are involved when some processing is related to class level but we need not involve the class or instances to perform any work. E.g. setting environment variables, counting no. of class instances or changing an attribute in other class.

```
class Myclass:
    n=0
    def __init__(self):
        Myclass.n=Myclass.n+1
    @staticmethod
    def noObjects():
        print('No. Of instances created:', Myclass.n)

obj1=Myclass()
obj2=Myclass()
obj3=Myclass()
Myclass.noObjects()
```

Output=>?

Passing members of one class to another(making use of static method):

```
class Emp:
        def  __init__(self, id, name, salary):
        self.id=id
        self.name=name
        self.salary=salary

    def display(self):
        print('id=',self.id)
        print('name=',self.name)
        print('salary=',self.salary)
Class Myclass:
    #method to receive Emp class instance and display
    @staticmethod
    def mymethod(e):
        e.salary+=1000;
        e.display()
e=Emp(10,'Raj Kumar',15000.75)
Myclass.mymethod(e)
```

So when to use static method??

It is when we want to pass some values from outside and perform some calculation in the method. Here, we are not touching class variables or instance variables.

When a processing is related to class but doesn’t need class or its instance to perform any work.

e.g. counting no of instance, changing attribute in another class, accept values n process them n return.

Try a program of Bank class where deposits and withdrawal are handled by using instance methods

bank_instance_method.py

Inner Class

Writing a class in another class is known as inner class or nested class.

```
class Person:
        def __init__(self):
                self.name=’Charles’
        def display(self):
                print(‘Name=’,self.name)
        class Dob:
                def __init__(self):
                        self.dd=10
                        self.mm=5
                        self.yy=1988
                def display(self):
                        print(‘Dob ={}/{}/{}’.format(self.dd,self.mm,self.yy))
p=Person()
p.display()
x=Person().Dob()
x.display()
print(x.yy)
```

Exception Handling

- Try
- Except
- Else
- Finally

EXCEPTION HANDLING

- Any abnormal condition in a program resulting to the disruption in the flow of the program.
- Whenever an exception occurs the program halts the execution.

- Thus exception is that error which python script is unable to tackle with.
- Exception in a code can also be handled

Hierarchy Of Exception:

- ZeroDivisionError: Occurs when a number is divided by zero.
- NameError: It occurs when a name is not found. It may be local or global.
- IndentationError: If incorrect indentation is given.
- IOError: It occurs when Input Output operation fails.
- EOFError: It occurs when end of file is reached and yet operations are being performed , etc..

List Of Standard Exceptions in Python

- List of standard exceptions available in Python.pdf

try:
    malicious code
except Exception1:
    execute code
except Exception2:
    execute code
....
....
except ExceptionN:
    execute code
else:
    In case of no exception, execute the **else** block code.

Here are few important points about the above-mentioned syntax −

- A single try statement can have multiple except statements. This is useful when the try block contains statements that may throw different types of exceptions.
- You can also provide a generic except clause, which handles any exception.
- After the except clause(s), you can include an else-clause. The code in the else-block executes if the code in the try: block does not raise an exception.
- The else-block is a good place for code that does not need the try: block's protection.

The try statement works as follows.

- First, the try clause (the statement(s) between the try and except keywords) is executed.

- If no exception occurs, the except clause is skipped and execution of the try statement is finished.

- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the try statement.

```
try:
    a=10/0
    print(a)
except ArithmeticError:
        print("This statement is raising an exception“)
else:
    print ("Welcome“)
```

Different Types Of Exception Handling Program

exception_types.py

```
try:
    a=10/0;
    print("Exception occurred“)
finally:
    print("Code to be executed“)
```

Output--

```
>>>
Code to be executed
Traceback (most recent call last):
  File "C:/Python27/noexception.py", line 2, in <module>
    a=10/0;
ZeroDivisionError: integer division or modulo by zero
>>>
```

- When an exception is thrown in the *try* block, the execution immediately passes to the *finally* block. After all the statements in the *finally* block are executed, the exception is raised again and is handled in the *except*statements if present in the next higher layer of the *try-except* statement.

The except Clause with Multiple Exceptions

You can also use the same except statement to handle multiple exceptions as follows −

```
try:
   You do your operations here;
   ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
   If there is any exception from the given exception list,
   then execute this block.
   ......................
else:
   If there is no exception then execute this block.
```

Exception Argument

- An exception can have an *argument*, which is a value that gives additional information about the problem.
- We can have a variable follow the name of the exception in the except statement.
- If trapping multiple exceptions-- variable follow the tuple of the exception.
- This variable receives the value of the exception mostly containing the cause of the exception. The variable can receive a single value or multiple values in the form of a tuple. This tuple usually contains the error string, the error number, and an error location.

Program for a single argument--
excp_arg.py

Raising an Exception

- raise keyword helps us to raise an exception
- Need to provide the name of the Exception class

```
try:
        raise ZeroDivisionError

except ZeroDivisionError:
        #handling code here
```

A sample program to see raise use:

raise1.py

raise is used to interrupt the code execution below the raise statement in try

It is more useful when we are dealing with user defined exceptions

i) To raise an exception, raise statement is used. It is followed by exception class name.
ii) Exception can be provided with a value that can be given in the parenthesis.
iii) To access the value "as" keyword is used. “ve" is used as a reference variable which stores the value of the exception.

Custom Exceptions

- create your own exceptions by deriving classes from the standard built-in exceptions.
- useful when you need to display more specific information when an exception is caught.

```
def validate(name):
        if len(name)<10:
                raise ValueError
```

Output:

>>>validate(‘joe’) gives ValueError

```
def validate(name):
        if len(name)<10:
                raise ValueError('name too short')
validate('joe')
```

Output will be ValueError: name too short

```
class NameTooShortError(ValueError):
        pass

def validate(name):
        if len(name)<10:
                raise NameTooShortError(name)
```

validate('joe') will give output as -- NameTooShortError:joe

Custom_exception.py

Custom_exception2.py

- Create a Program teacher.py containing Class Teacher and its methods – setid, setname, setaddress and getid, getname, getaddress.

- Create another program which imports teacher module and class.

- Create an object of Teacher class here and call the respective methods and print the values

Exercise 2:

Create a program which demonstrates the concepts such as multilevel inheritance, multiple inheritance, interface, class methods, static methods and inner class and any one exception handling mechanism(system defined or user defined).

- Another way how we can work for it is that inherit the class Teacher in Student.

- class Student(Teacher)

- Create methods which are specific for students ie. Marks and rest can be called straight from the students class.

Some questions here?

- We don’t create an object on Parent class still use its members. How?
- Why go for inheritance?
- If I create an object of parent class then?
- So which class object do we generally create?

Now let us see Constructors in Inheritance

- Create a class Father with instance variable property= 8000 in constructor
- and display it using method display_property

- Create a class Son inherited from Father, write pass, create its object and call display_property method.

- What it says about constructors in python?

- Now create a constructor in son class as well.
- Assign instance variable valeu with self.property as 2000 and then call display_property method using son class object.
- This is known as constructor overriding.
- Now try adding a method in child class with same name as of parent.
- This is method overriding.

- So can we still make use of parent constructor and method?
- Who can help?

- Super method!