Python Programming

EDUCATION FOR PEACE

(ESTD-1995)

RGM College of Engineering & Technology (Autonomous)

Department of Computer Science & Engineering

AY: 2021-2022

PYTHON’S OBJECT ORIENTED PROGRAMMING - 2

Guido Van Rossum

Agenda:

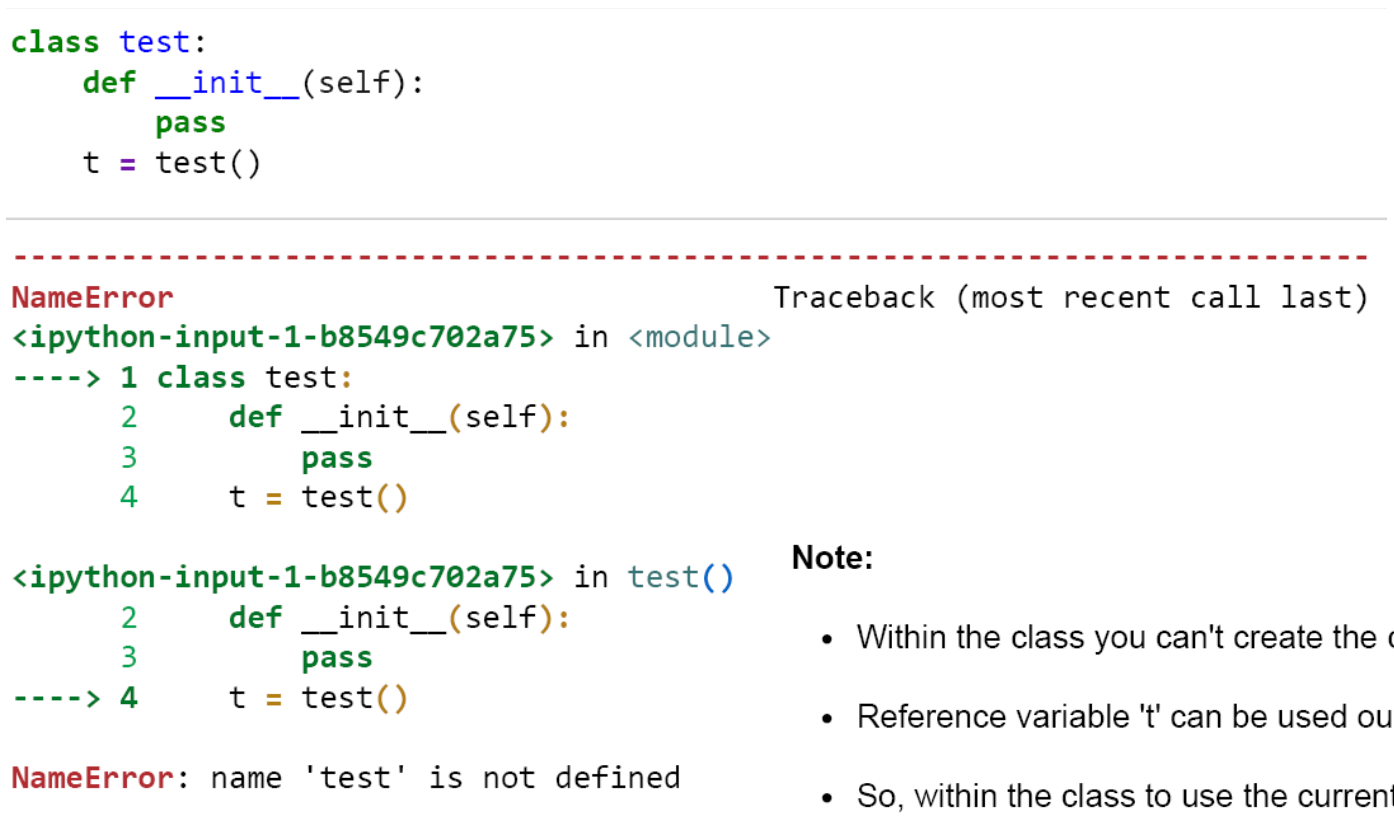
1. The Complete Post-mortem of ‘self’ variable

2. The Complete Story of Python Constructors

1. The Complete Post-mortem of 'self' variable

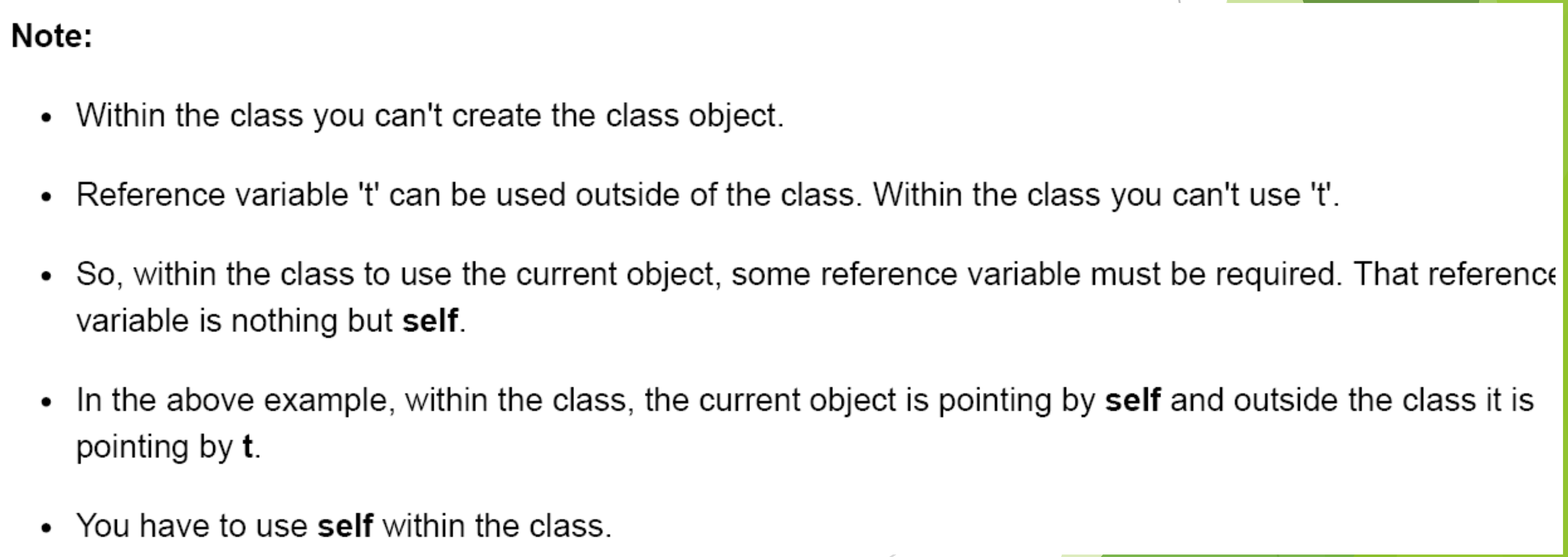
- **self** is the most commonly used variable in our python programs.
- **self** is a reference variable, which is always pointing to the current object.
- Within the Python class if you want to access (or) refer the current object, some reference variable must be used. That reference variable itself is nothing but **self**.

Consider the following example,



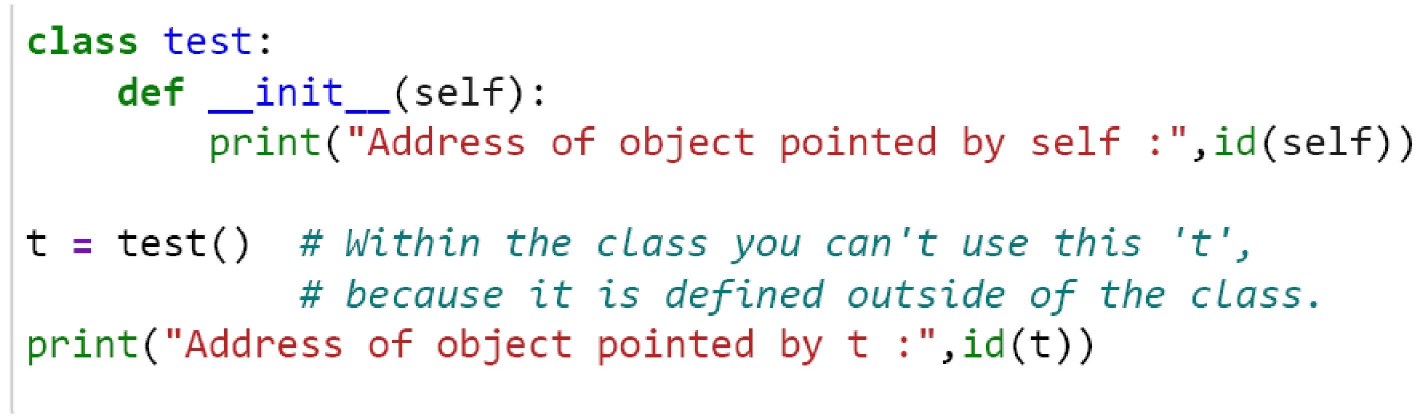
```
class test:
    def __init__(self):
        pass
    t = test()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-b8549c702a75> in <module>
----> 1 class test:
      2     def __init__(self):
      3         pass
      4     t = test()

<ipython-input-1-b8549c702a75> in test()
      2     def __init__(self):
      3         pass
----> 4     t = test()

NameError: name 'test' is not defined
```



Note:

- Within the class you can't create the class object.
- Reference variable 't' can be used outside of the class. Within the class you can't use 't'.
- So, within the class to use the current object, some reference variable must be required. That reference variable is nothing but **self**.
- In the above example, within the class, the current object is pointing by **self** and outside the class it is pointing by **t**.
- You have to use **self** within the class.



```python
class test:
    def __init__(self):
        print("Address of object pointed by self :",id(self))

t = test()  # Within the class you can't use this 't',
            # because it is defined outside of the class.
print("Address of object pointed by t :",id(t))
```

Address of object pointed by **self** : 1768116875376
Address of object pointed by **t** : 1768116875376

Note:

In the above example, both **t** and **self** are pointing to the same object.

Refer the following examples to understand about the **self** variable:

I.

```python
class test:
    def __init__(self):
        print("Constructor")

    def m1(self,x):
        print("x value : ",x)

t = test()
t.m1(55)
```

```
Constructor
x value :  55
```

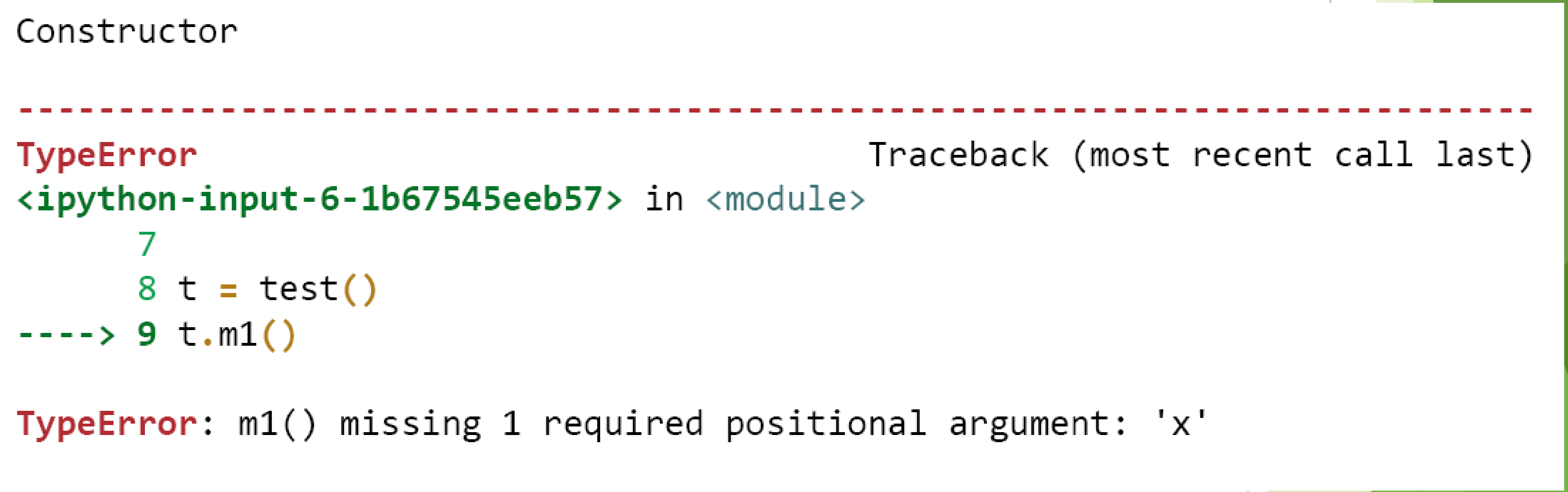
II.

```
class test:
    def __init__(self):
        print("Constructor")

    def m1(self,x):
        print("x value : ",x)

t = test()
t.m1('RGM')
```

```
Constructor
x value :  RGM
```

III.

```python
class test:
    def __init__(self):
        print("Constructor")

    def m1(self,x):
        print("x value : ",x)

t = test()
t.m1()
```



```
Constructor

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-1b67545eeb57> in <module>
      7
      8 t = test()
----> 9 t.m1()

TypeError: m1() missing 1 required positional argument: 'x'
```

Another Example:

```
class student:
    def __init__(self,name,rollno,marks):
        self.name = name
        self.rollno = rollno
        self.marks = marks

    def talk(self):
        print("Hello I am :",self.name)
        print("My Roll number is :",self.rollno)
        print("My Marks are :",self.marks)

s1 = student("Karthi",101,67)
s1.talk()
```

```
Hello I am : Karthi
My Roll number is : 101
My Marks are : 67
```

Points to Ponder:

1. The first argument to the constructor (i.e., init()) is self.
2. The first argument of the instance method is self.
3. We are not responsible to pass the value to the self. PVM is responsible to perform that task.

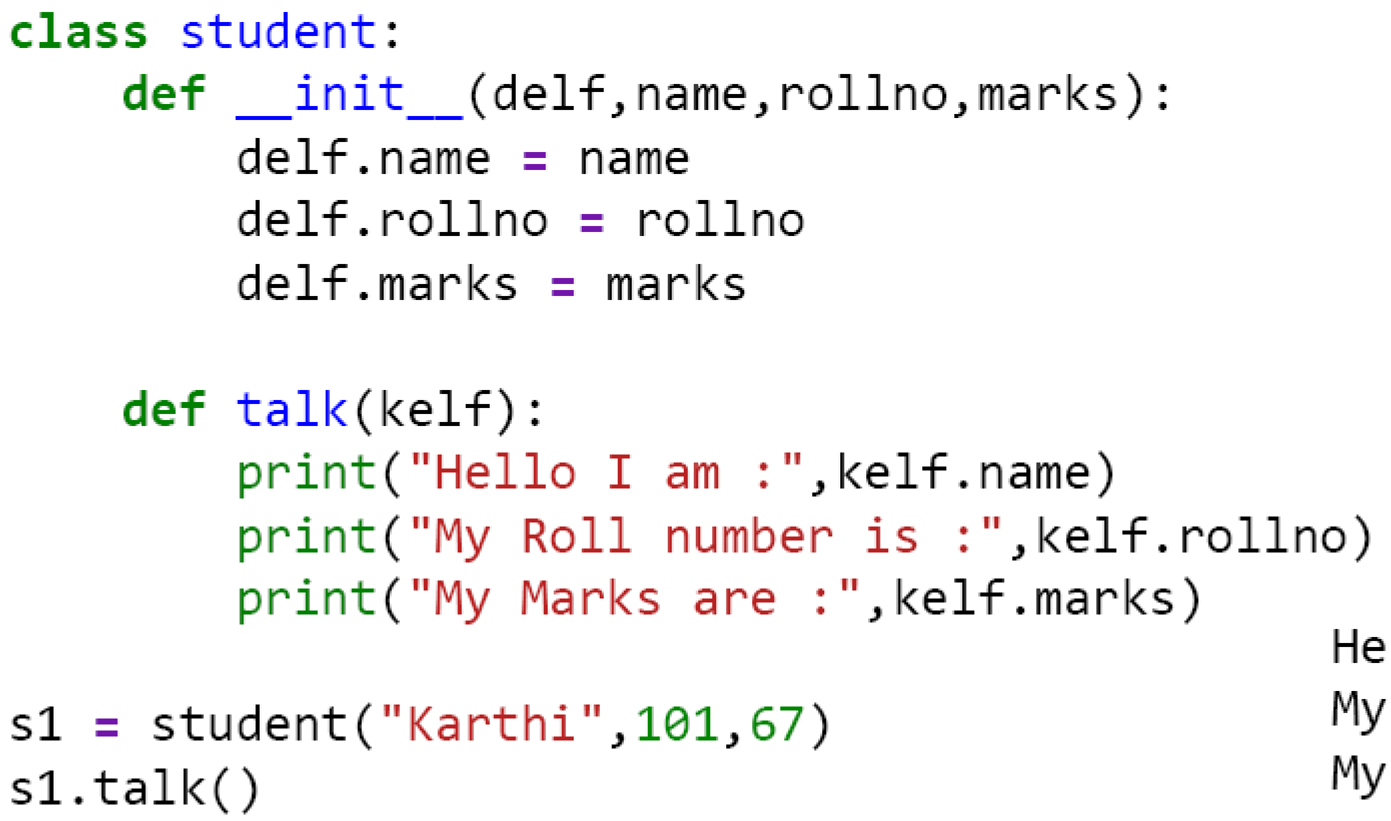
Q. For which purpose variable 'self' is used within the class?

Ans:

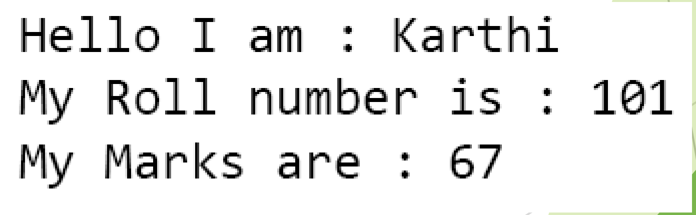
1. To declare Instance variables (i.e.,name, rollno and marks).
2. To access the values of Instance variables.

Is 'self' a keyword in Python?

See the below example, where we are using 'delf' and 'kelf' instead of 'self'.



```python
class student:
    def __init__(delf,name,rollno,marks):
        delf.name = name
        delf.rollno = rollno
        delf.marks = marks

    def talk(kelf):
        print("Hello I am :",kelf.name)
        print("My Roll number is :",kelf.rollno)
        print("My Marks are :",kelf.marks)

s1 = student("Karthi",101,67)
s1.talk()
```



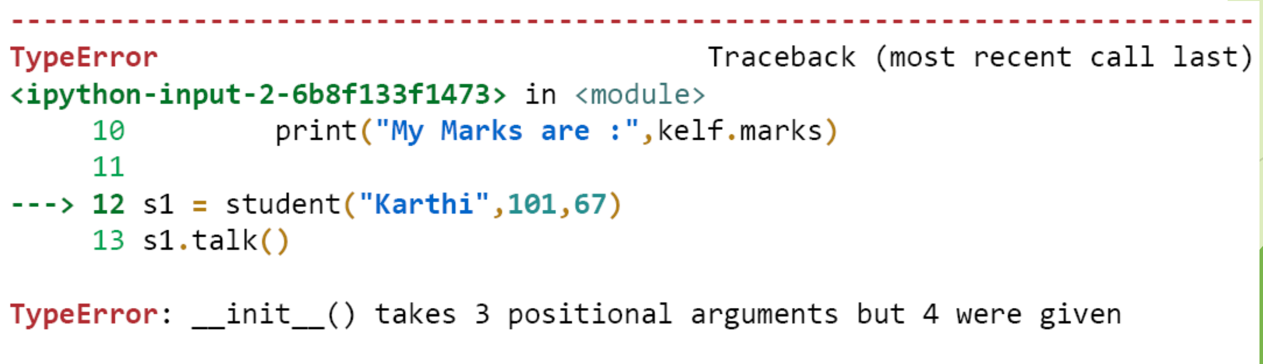
```
Hello I am : Karthi
My Roll number is : 101
My Marks are : 67
```

Note:

- self is not a keyword in python. You can use either 'delf' or 'kelf' instead of 'self'.
- But, it is not recommended. Universal accepted terminology is always 'self'.
- The first argument is implicit variable pointing to current object. Value to this argument passed by the PVM.
- Name of the first argument is not important.

Eg:

```
class student:
    def __init__(name,rollno,marks):
        delf.name = name
        delf.rollno = rollno
        delf.marks = marks

    def talk(kelf):
        print("Hello I am :",kelf.name)
        print("My Roll number is :",kelf.rollno)
        print("My Marks are :",kelf.marks)

s1 = student("Karthi",101,67)  # 3+ 1 value provides PVM for self
s1.talk()
```



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-6b8f133f1473> in <module>
     10         print("My Marks are :",kelf.marks)
     11 
---> 12 s1 = student("Karthi",101,67)
     13 s1.talk()

TypeError: __init__() takes 3 positional arguments but 4 were given
```

self variable conclusions:

1. Within the python class to refer current object, some variable must be required which is nothing but **self.**
2. self is a reference variable always pointing to current object.
3. The first argument to the constructor and instance method is always self.
4. At the time of calling constructor or instance method, we are not required pass any value to self variable. Internally PVM is responsible to provide value.
5. The main purpose of self variable within the class is to declare instance variables and to access the value of instance variables.
6. self is not a keyword and instead of 'self' we can use *delf* or *kelf* etc. But recommended to use self.
7. We can not use self from outside of the class.

2. The Complete Story of Python Constructors

Features of Constructor:

1. Constructor is a special method in python.
2. The name of the constructor should be _ _ **init _ _(self).**
3. In Java, the name of the constructor is the name of the class. But in Python, it is not varied from class to class.
4. The constructor name in Python is always fixed (i.e.,_ _ init _ _).
5. We are not required to call constructor explicitly. Whenever we are creating an object, automatically constructor will be executed.
6. Based on our requirement, we can call constructor explicitly, then it will be executed just like a normal method.

7. Per object constructor will be executed only once.

8. Constructor can take at least one argument(at least self).

9. The main purpose of constructor is to declare and initialize instance variables.

10. Within the class, Constructor is optional and if we are not providing any constructor then python will provide default constructor.

11. In Python, overloading concept is not applicable for constructors. Hence we cannot define multiple constructors within the same class. If we are trying to define multiple constructors, only last constructor be considered.

Example Programs on Constructors:

I.

```
class test:
    def __init__(self):                   # Constructor method
        print("Constructor execution")
t1=test()
```

```
Constructor execution
```

II.

```
class test:
    def __init__(self):                   # Constructor method
        print("Constructor execution")
t1=test()
t2=test()
```

```
Constructor execution
Constructor execution
```

III.

```
class test:
    def __init__(self):                  # Constructor method
        print("Constructor execution")
t1=test()
t2=test()
t3=test()    # 3 times _ _ init() _ _ will be executed.
```

```
Constructor execution
Constructor execution
Constructor execution
```

Note:

- **The main purpose of constructor is to declare and initialize instance variables.**

Another Example:

I.

```python
class student:
    def __init__(self,name,rollno,marks):
        self.name = name
        self.rollno = rollno      # Instance variables creation
        self.marks = marks

s1 = student("Sourav",102,98)
s2 = student("Sachin", 103,95)
print(s1.name,s1.rollno,s1.marks)
print(s2.name,s2.rollno,s2.marks)
```

```
Sourav 102 98
Sachin 103 95
```

Another Example:

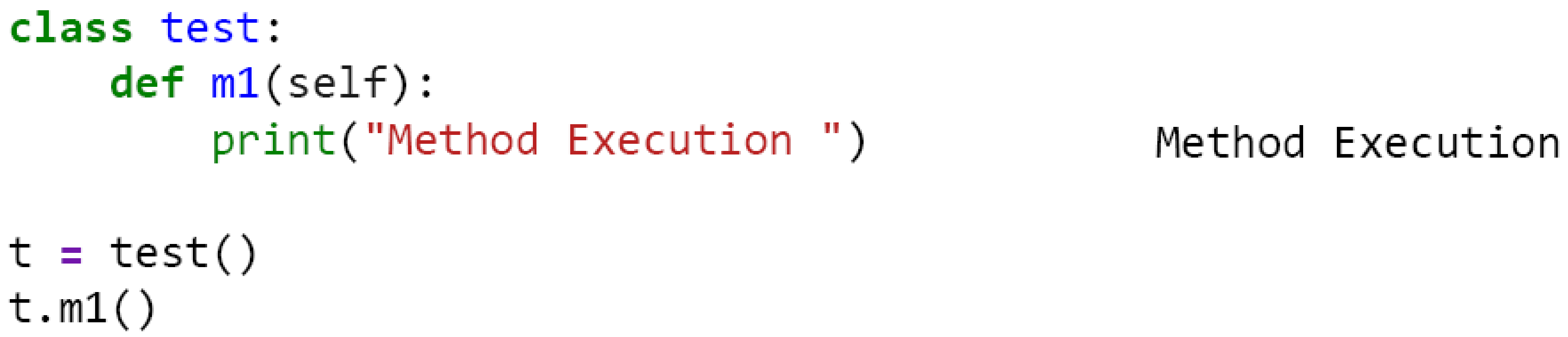
I.

```
class Student:
    ''''' This is student class with required data'''
    def __init__(self,x,y,z):
        self.name=x
        self.rollno=y
        self.marks=z

    def display(self):
        print("Student Name:{}\nRollno:{} \nMarks:{}".format(self.name,self.rollno,self.marks))

s1=Student("Virat",101,80)
s1.display()
s2=Student("Rohit",102,100)
s2.display()
```

```
Student Name:Virat
Rollno:101
Marks:80
Student Name:Rohit
Rollno:102
Marks:100
```

- Constructor is optional and if we are not providing any constructor then python will provide default constructor.
- Default constructor won't perform anything, but for syntactical purpose python provides default constructor.

Eg:



```python
class test:
    def m1(self):
        print("Method Execution ")

t = test()
t.m1()
```

```
Method Execution
```

- Based on our requirement, We can call constructor explicitly. Then it will be executed just like normal method. New object won't be created.

Eg:

```
class test:
    def __init__(self):
        print("Constructor Execution ")
t = test()
t.__init__()  # Calling constructor explicitly
t.__init__()  # Only one object created
t.__init__()
```

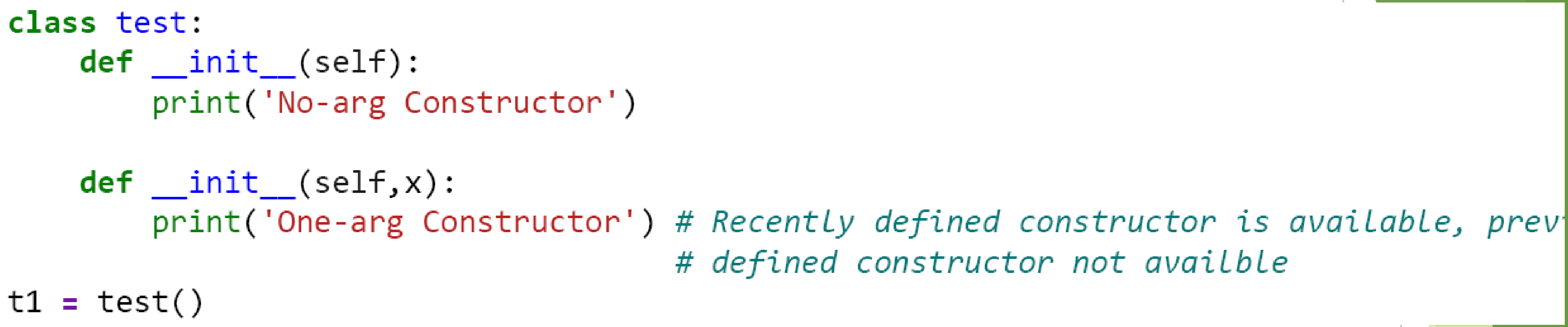
```
Constructor Execution
Constructor Execution
Constructor Execution
Constructor Execution
```

Key Points:

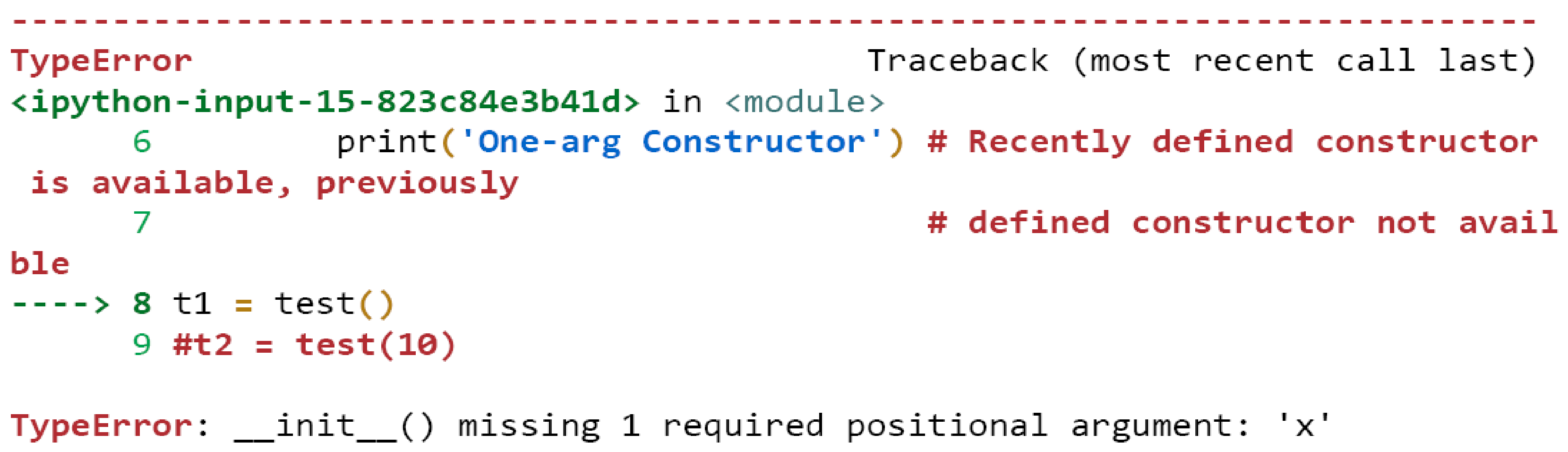
- If two methods with same name, but different arguments, such type of methods known as overloaded methods.
- In Python, method overloading concept is not supported.
- In Python, constructor overloading is not applicable.
- In Python, multiple constructors can define within a class, but PVM is going to consider only the last constructor.

Eg:

I.

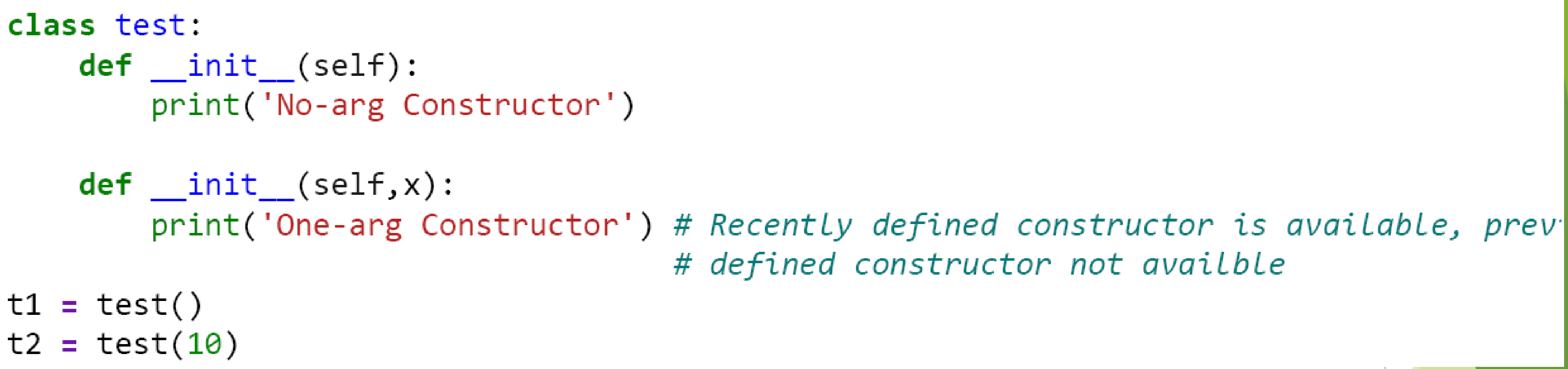


```
class test:
    def __init__(self):
        print('No-arg Constructor')

    def __init__(self,x):
        print('One-arg Constructor') # Recently defined constructor is available, prev
                                     # defined constructor not availble

t1 = test()
```

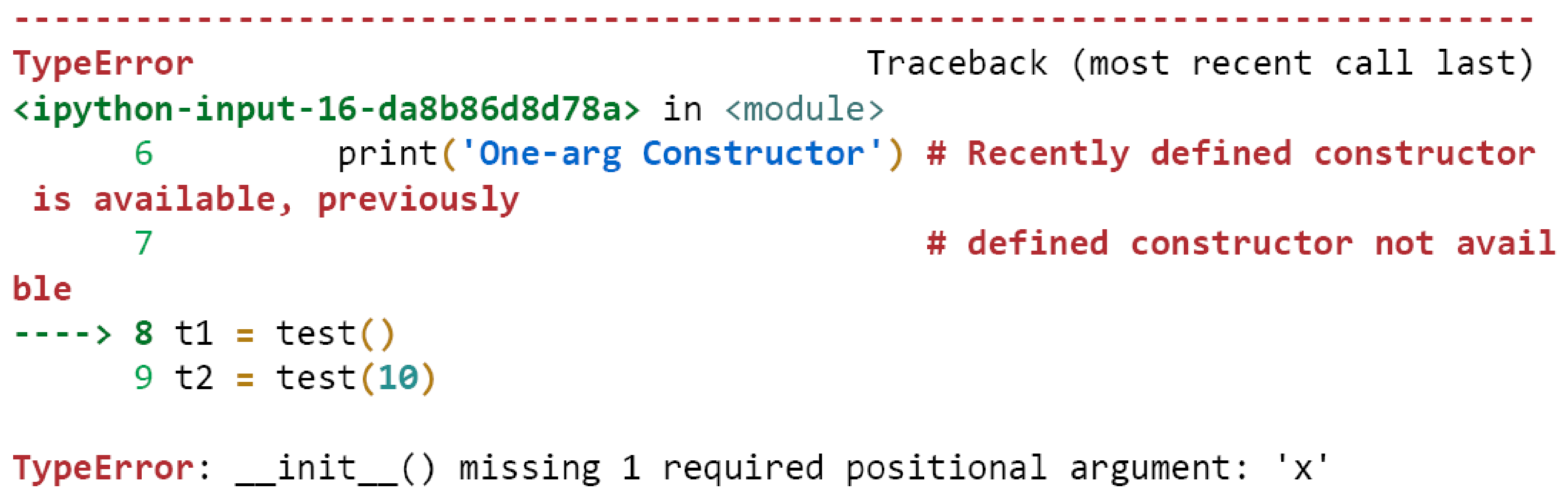


```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-15-823c84e3b41d> in <module>
      6         print('One-arg Constructor') # Recently defined constructor
 is available, previously
      7                                      # defined constructor not avail
ble
----> 8 t1 = test()
      9 #t2 = test(10)

TypeError: __init__() missing 1 required positional argument: 'x'
```

II.



```
class test:
    def __init__(self):
        print('No-arg Constructor')

    def __init__(self,x):
        print('One-arg Constructor') # Recently defined constructor is available, prev
                                     # defined constructor not availble

t1 = test()
t2 = test(10)
```

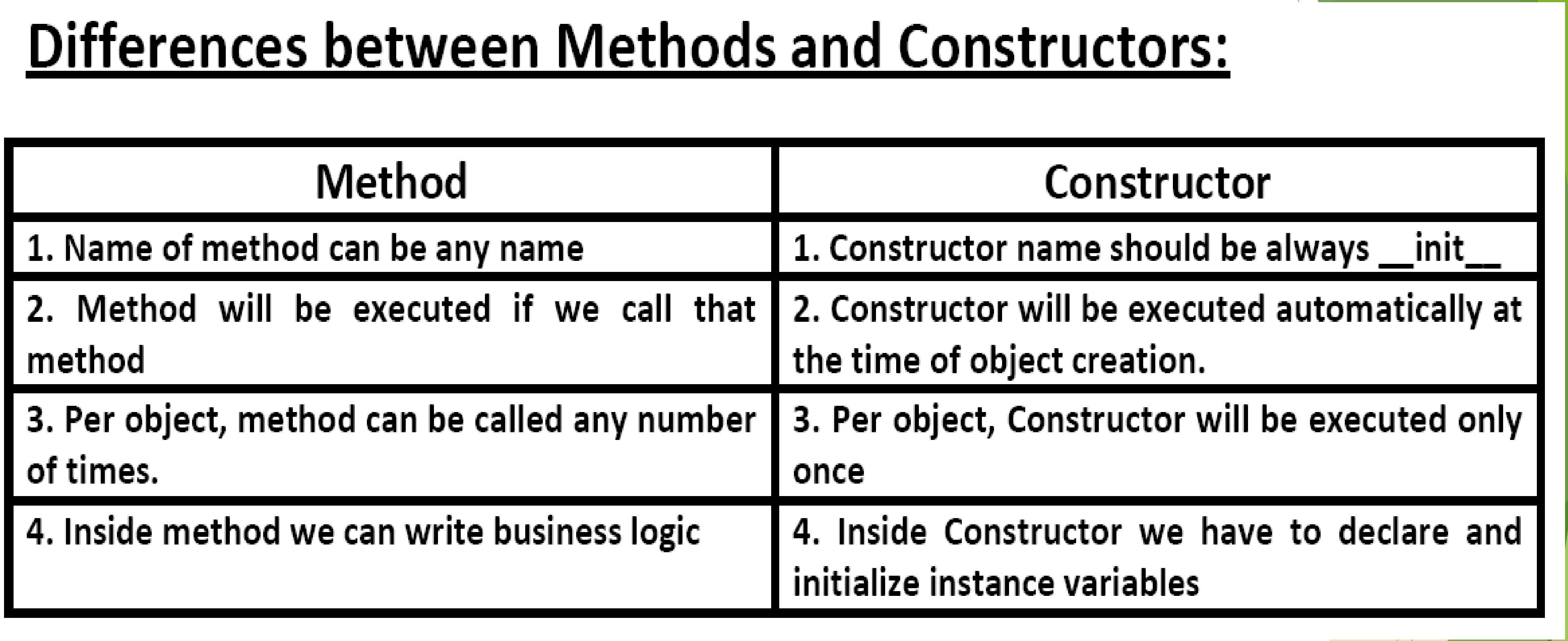


```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-16-da8b86d8d78a> in <module>
      6         print('One-arg Constructor') # Recently defined constructor
 is available, previously
      7                                      # defined constructor not avail
ble
----> 8 t1 = test()
      9 t2 = test(10)

TypeError: __init__() missing 1 required positional argument: 'x'
```

III.

```
class test:
    def __init__(self):
        print('No-arg Constructor')

    def __init__(self,x):
        print('One-arg Constructor') # recent

#t1 = test()
t2 = test(10)
```

```
 One-arg Constructor
```



Differences between Methods and Constructors:

Method	Constructor
1. Name of method can be any name	1. Constructor name should be always __init__
2. Method will be executed if we call that method	2. Constructor will be executed automatically at the time of object creation.
3. Per object, method can be called any number of times.	3. Per object, Constructor will be executed only once
4. Inside method we can write business logic	4. Inside Constructor we have to declare and initialize instance variables

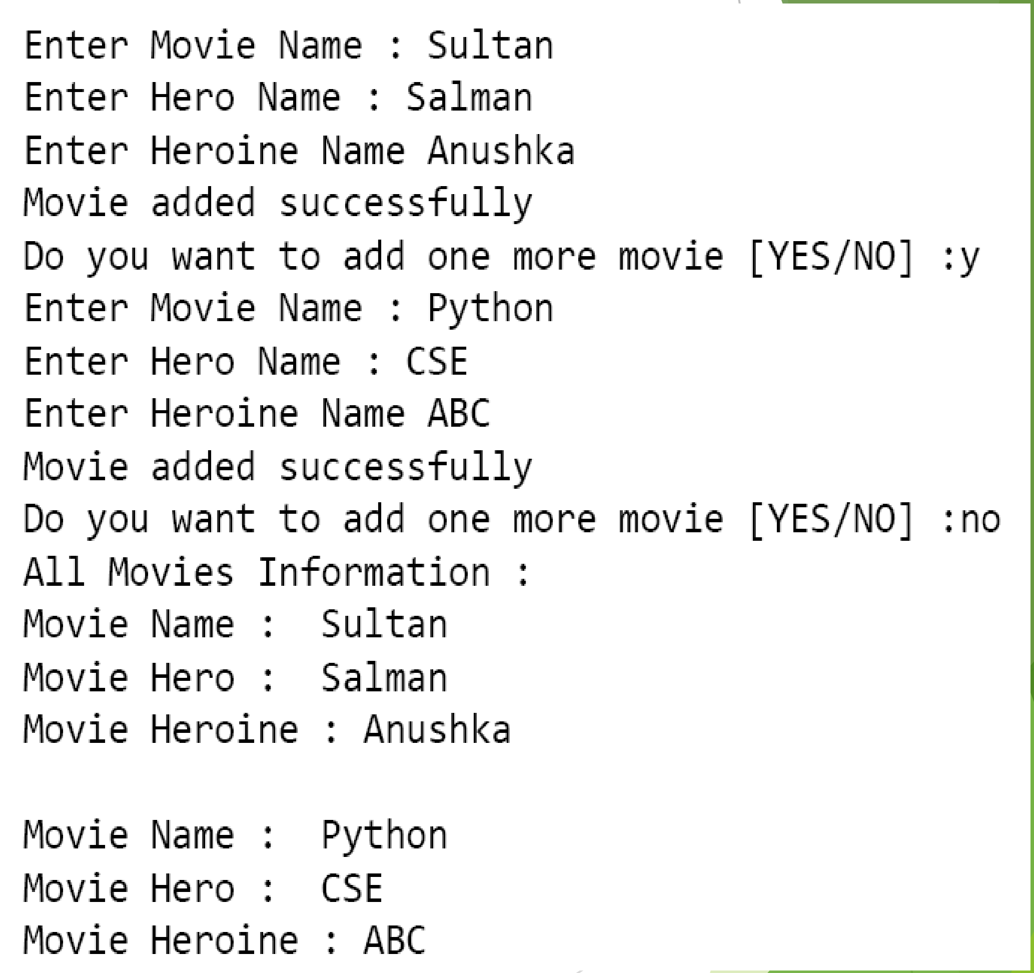
Mini Application to Understand Object Oriented Programming

```python
class Movie:

    ''' This Class developed by CSE II Year Students for Demo'''

    def __init__(self,title,hero,heroine):
        # Here, title,hero.heroine are Local variables to hold provided values
        # at the time of creating objectcs
        self.title = title
        self.hero = hero # self.title,self.hero,self.heroine are instance variables
        self.heroine = heroine

    def info(self):
        print("Movie Name : ",self.title)
        print("Movie Hero : ",self.hero)
        print("Movie Heroine :",self.heroine)

list_of_movies = []
while True:
    title = input("Enter Movie Name : ")
    hero = input("Enter Hero Name : ")
    heroine = input("Enter Heroine Name " )
    m = Movie(title,hero,heroine)    # Creation of Movie object
    list_of_movies.append(m)         # Adding of movie object to the List
    print("Movie added successfully ")

    option = input("Do you want to add one more movie [YES/NO] :")

    if option.lower() == 'no':
        break

print("All Movies Information : ")

for movie in list_of_movies:
    movie.info()                     # Accesing the movies information
    print()
```

```
Enter Movie Name : Gundamma Katha
Enter Hero Name : NTR
Enter Heroine Name savithri
Movie added successfully
Do you want to add one more movie [YES/NO] :YES
Enter Movie Name : Gundamma Katha
Enter Hero Name : ANR
Enter Heroine Name Jamuna
Movie added successfully
Do you want to add one more movie [YES/NO] :NO
All Movies Information :
Movie Name :  Gundamma Katha
Movie Hero :  NTR
Movie Heroine : savithri

Movie Name :  Gundamma Katha
Movie Hero :  ANR
Movie Heroine : Jamuna
```

II.

```python
class Movie:

    ''' This Class developed by CSE II Year Students for Demo'''

    def __init__(self,a,b,c):
        # Here, title,hero.heroine are Local variables to hold provided values
        # at the time of creating objectcs. These are temporary variables.
        self.title = a
        self.hero = b # self.title,self.hero,self.heroine are instance variabl
        self.heroine = c   # These are standard variables.

    def info(self):
        print("Movie Name : ",self.title)
        print("Movie Hero : ",self.hero)
        print("Movie Heroine :",self.heroine)

list_of_movies = []
while True:
    title = input("Enter Movie Name : ")
    hero = input("Enter Hero Name : ")
    heroine = input("Enter Heroine Name " )
    m = Movie(title,hero,heroine)    # Creation of Movie object
    list_of_movies.append(m)         # Adding of movie object to the list
    print("Movie added successfully ")

    option = input("Do you want to add one more movie [YES/NO] :")

    if option.lower() == 'no':
        break

print("All Movies Information : ")

for movie in list_of_movies:
    movie.info()                   # Accesing the movies information
    print()
```



```
Enter Movie Name : Sultan
Enter Hero Name : Salman
Enter Heroine Name Anushka
Movie added successfully
Do you want to add one more movie [YES/NO] :y
Enter Movie Name : Python
Enter Hero Name : CSE
Enter Heroine Name ABC
Movie added successfully
Do you want to add one more movie [YES/NO] :no
All Movies Information :
Movie Name :  Sultan
Movie Hero :  Salman
Movie Heroine : Anushka

Movie Name :  Python
Movie Hero :  CSE
Movie Heroine : ABC
```

Conclusions:

The above application demonstrates about the following:

- How you can define a Class?
- How you can create an Object with dynamic information and how to use that object?

Note:

Q. Once the program execution is over. Is this movies data stored in the list is available or not?

Ans: All the data stored in the objects is stored inside the heap memory. Once the program execution completed, PVM will be shutdown. Before shutdown PVM, heap area will be cleared. Then automatically the objects will be gone. So, while executing program, if you want to store data, then you have to go for either files concept or databases concept.

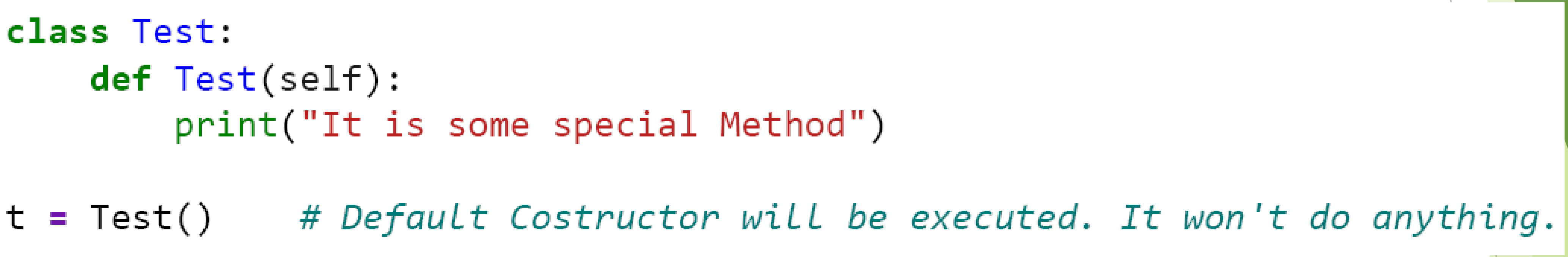
[Files and databases are permanent storage areas]

[Lists, tuples and dictionaries etc., are temporary storage areas]

Q. What happens if we provide a method name same as class name?

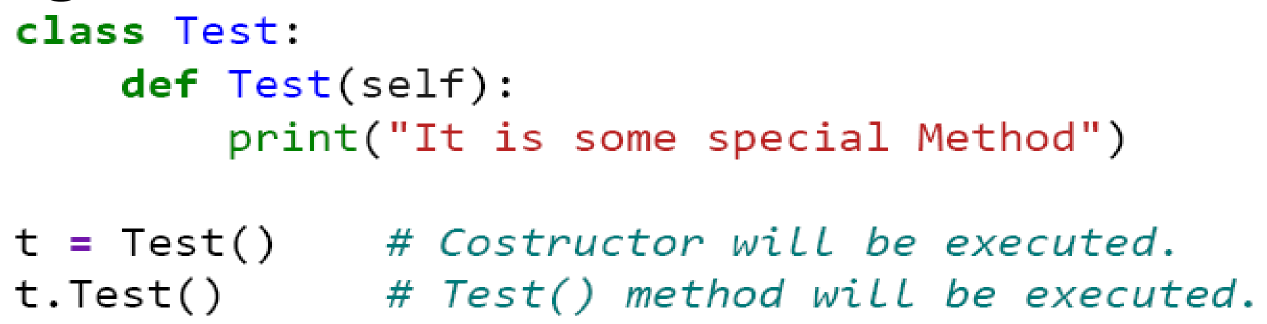
Ans: No Problem, it is acceptable. We can use method name as class name but not recommended.

Eg 1:

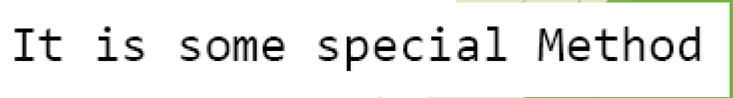


```
class Test:
    def Test(self):
        print("It is some special Method")

t = Test()    # Default Costructor will be executed. It won't do anything.
```

Eg 2:



```
class Test:
    def Test(self):
        print("It is some special Method")

t = Test()     # Costructor will be executed.
t.Test()       # Test() method will be executed.
```



```
It is some special Method
```

Any question?

If you try to practice programs yourself, then you will learn many things automatically

Spend few minutes and then enjoy the study

Thank You