Tutorial - 10

TA: Nandika Jain

INDRAPRASTHA INSTITUTE *of* INFORMATION TECHNOLOGY **DELHI**

What is modularity?

- Dividing a program into logical parts, with each part doing some well defined computation or providing defined interface and functionality (like use of functions.)
- Cannot write a program of thousands of lines as just one program - must break it into parts and develop parts separately. (**Reusability of code**)
- Easier to debug your code as well.
- Classes and objects is another way to provide modularity, capability to logically divide a problem into computational units. (discussed in the later part of the slides.)

Data Types

- **Basic Data Types:** int, float, bool

  On objects of these types, we can do the operations defined on them,

  e.g. +, -, * etc (on int, float), and not/or/and for bool
- **Complex data types:** lists, sets, dictionaries, tuples

  We can define variables of this type, and perform the operations on

  them.

Data Structures

- A data structure is a particular way of organizing data in a computer so that it can be used effectively.

  For eg. Storing marks of students, instead of multiple ints, we are storing them in a list.

  Stack (LIFO), Linked List, Queue (FIFO)

- Object oriented constructs provide this capability - define new user defined data types.

Object Oriented Programming

- Ability to define a new type: the structure of an object of this type i.e. what

attributes/variables it has, and the operations defined on objects of this type.
(Ability to define variables of this type, and invoke operations on it)

- Can define a "car" as a type (say for a game) which has ops like start, accelerate, turn left, turn right, ... and attributes like color, make, current speed, weight, ...

Creating a class:

- Python allows defining a new type as a class.
- Class is like an object constructor, or a "blueprint" for creating objects.
- **class** myClass:
            pass

Here, we use the class keyword to define an empty class myClass. From class, we construct instances. An instance is a specific object created from a particular class.

Object

- An object (instance) is an instantiation of a class.
- We can create objects of myClass type and assign them to variables -as with language defined types like dict, sets, lists, int.
- ***p1 = MyClass()***

  Role of variable is same as with all types; the nature of object is now of myClass type.

Methods & Attributes

- A class also has methods - functions which define the operations on this new type.
- Only these methods can be executed on objects of this class
- A class has attributes - these are variables in its scope, accessed from within the class to implement the methods of the class.
- Attributes define the state of an object of the class

Identify Methods and Attributes

The data values which we store inside an object are called attributes, and the functions which are associated with the object are called methods.

```
class Person:

    def __init__(self, name, surname, birthdate, address, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate

        self.address = address
        self.telephone = telephone
        self.email = email

    def age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year

        if today < datetime.date(today.year, self.birthdate.month, self.birthdate.day):
            age -= 1

        return age
```

__init__

- All classes have a function called __init__(), which is always executed when the class is being initiated.
- If you don’t define it, then: It inherits an __init__ that takes no arguments other than self and does nothing.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

Question!!

Create a class named Person, use the __init__() function to assign values for name and age:

Solution:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

Calling Methods

- A class has no state, it is just a definition - the state is within objects.
- When a method is invoked on an object, the function defined in class executes on the state of the object on which it is invoked.
- Operations on an object obj are invoked using dot (.) notation:

  object_variable.operation (args)

self

- In the method, it is the first parameter self - the ref to the object
- All attributes of an object are accessed as self.attribute
- The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class.

Execution

1. When a program is executed, the class definitions are noted by interpreter, but no method is looked at.
2. When an object is created, then the __init__() method is visited and executed.
3. When a method is executed on an object, the method function is executed with the ref to the object as the first param.

```
<MyClass_obj1>.<method_1>(args)
```

The above statement calls the method_1 defined in MyClass with the reference of MyClass_obj1 in self

Modifying State

Set the age of p1 to 40:

```
p1.age = 40
```

Practice Question 1

Design a class called Date. The class should store a date in three integers: month, day, year to initialise.

1) Define getters and setters for this class.

Eg. methods getmonth(), setday(d) etc.

2) There should be methods to print the date in the following forms:

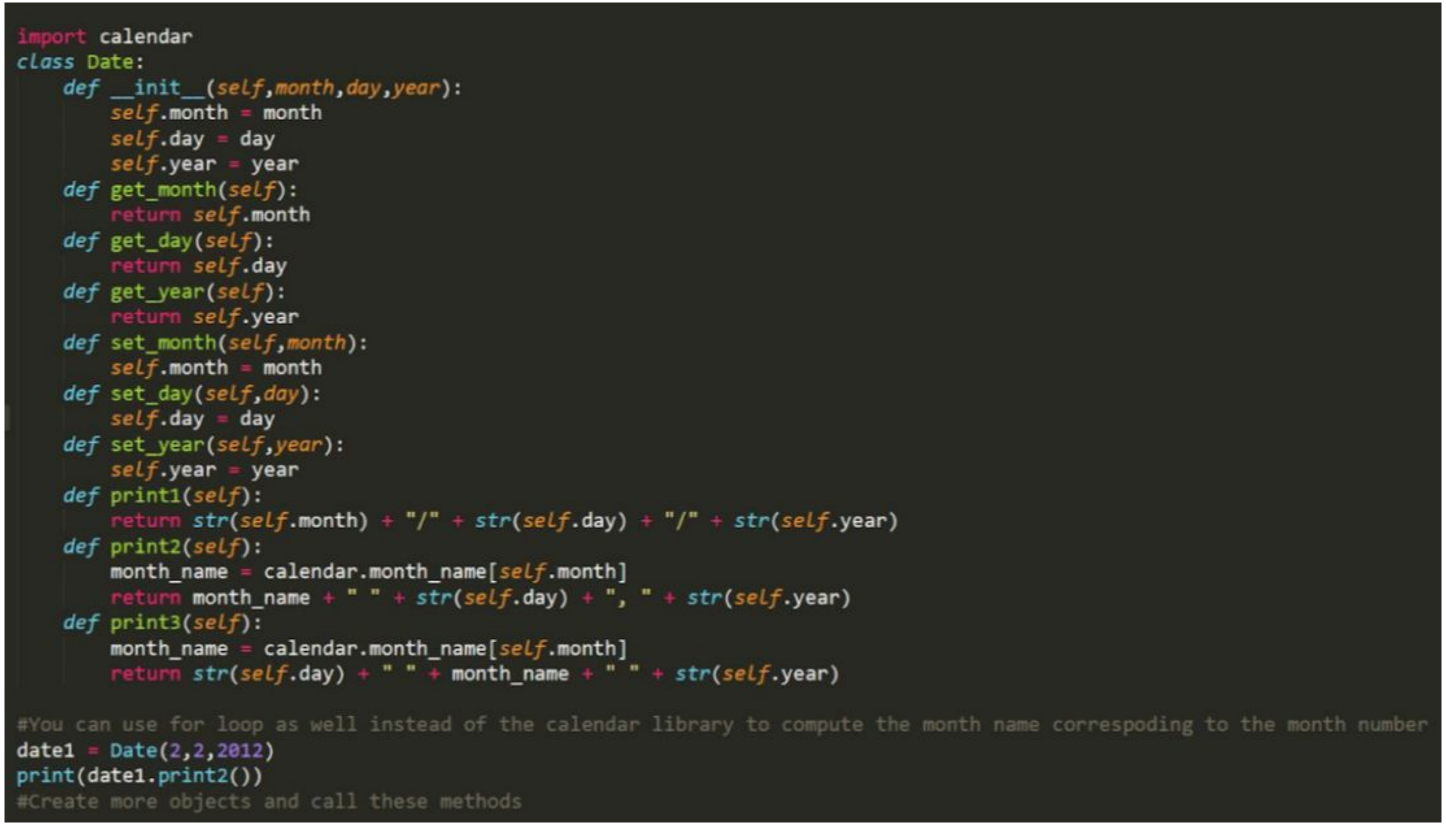
2/23/2012

February 23, 2012

23 February 2012

3) Create objects of this class and call these methods.

Solution to Practice Question 1



```python
import calendar
class Date:
    def __init__(self,month,day,year):
        self.month = month
        self.day = day
        self.year = year
    def get_month(self):
        return self.month
    def get_day(self):
        return self.day
    def get_year(self):
        return self.year
    def set_month(self,month):
        self.month = month
    def set_day(self,day):
        self.day = day
    def set_year(self,year):
        self.year = year
    def print1(self):
        return str(self.month) + "/" + str(self.day) + "/" + str(self.year)
    def print2(self):
        month_name = calendar.month_name[self.month]
        return month_name + " " + str(self.day) + ", " + str(self.year)
    def print3(self):
        month_name = calendar.month_name[self.month]
        return str(self.day) + " " + month_name + " " + str(self.year)

#You can use for loop as well instead of the calendar library to compute the month name correspoding to the month number
date1 = Date(2,2,2012)
print(date1.print2())
#Create more objects and call these methods
```

Practice Question 2 (Homework)

Define a Movie class that takes Movie Title, Director Name, Genre and IMDB rating to initialize.

1) You may define appropriate getters and setters.

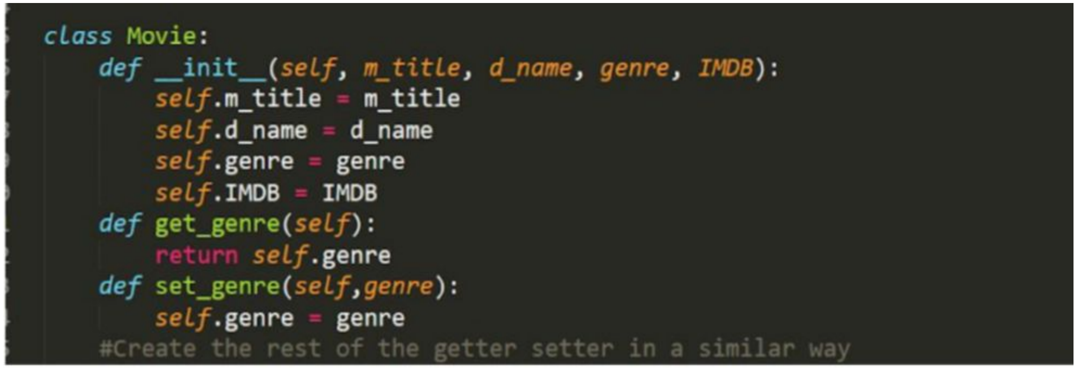
2) Create multiple objects of this class(atleast 5)

3) Run the following queries on these objects:

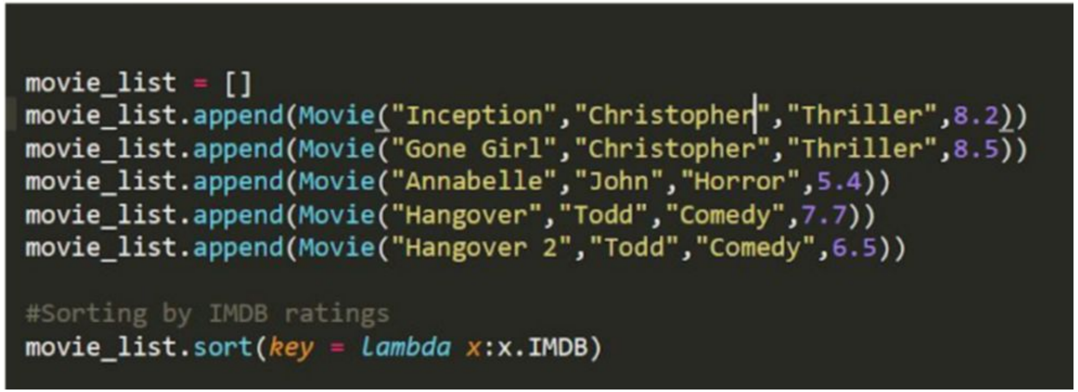
- Sort Movies by their IMDB rating, in ascending order (You may use the inbuilt functions)
- List Title of Movies with IMDB rating above 7.0
- List Title of Movies directed by a particular director
- List Directors directing a particular genre

Link to sort based on ratings:
https://stackoverflow.com/questions/3766633/how-to-sort-with-lambda-in-python

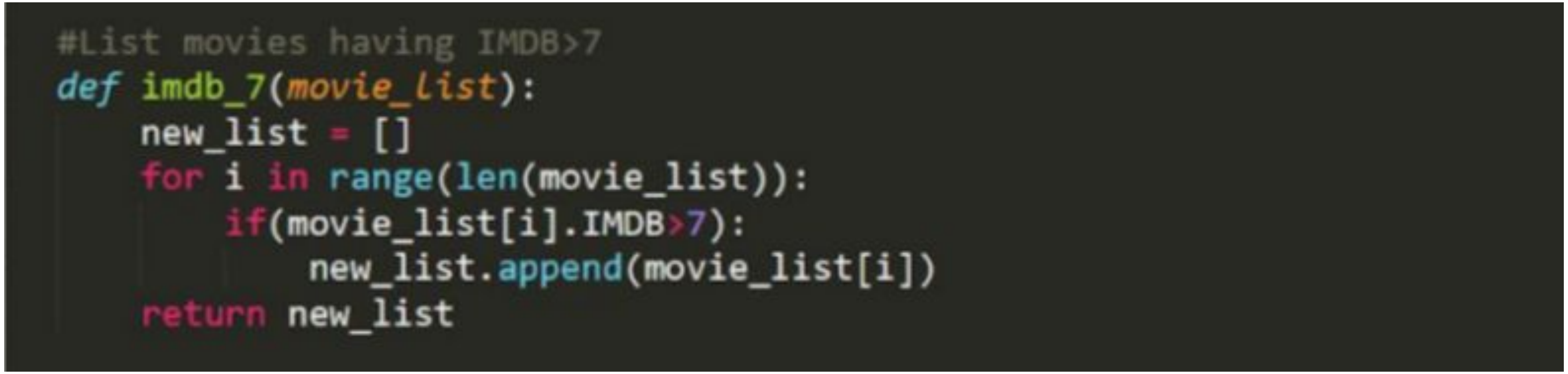
Solution to Practice Question 2



```python
class Movie:
    def __init__(self, m_title, d_name, genre, IMDB):
        self.m_title = m_title
        self.d_name = d_name
        self.genre = genre
        self.IMDB = IMDB
    def get_genre(self):
        return self.genre
    def set_genre(self,genre):
        self.genre = genre
    #Create the rest of the getter setter in a similar way
```



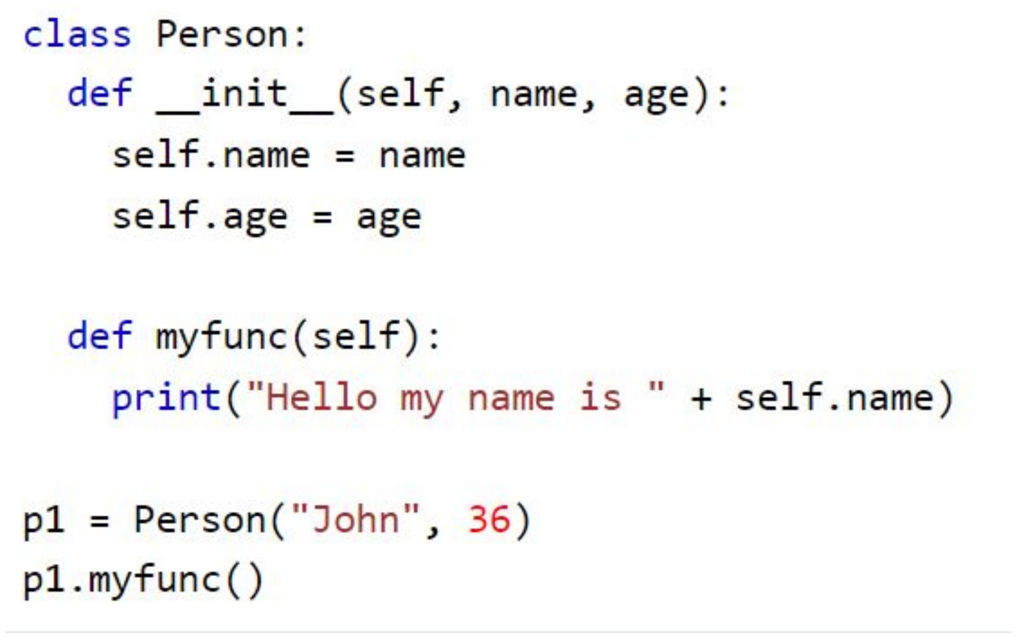
```python
movie_list = []
movie_list.append(Movie("Inception","Christopher","Thriller",8.2))
movie_list.append(Movie("Gone Girl","Christopher","Thriller",8.5))
movie_list.append(Movie("Annabelle","John","Horror",5.4))
movie_list.append(Movie("Hangover","Todd","Comedy",7.7))
movie_list.append(Movie("Hangover 2","Todd","Comedy",6.5))

#Sorting by IMDB ratings
movie_list.sort(key = lambda x:x.IMDB)
```


```
#List movies having IMDB>7
def imdb_7(movie_list):
    new_list = []
    for i in range(len(movie_list)):
        if(movie_list[i].IMDB>7):
            new_list.append(movie_list[i])
    return new_list
```

Similarly, write the other 2 functions.

Practice Question 3

Find the output of the code:



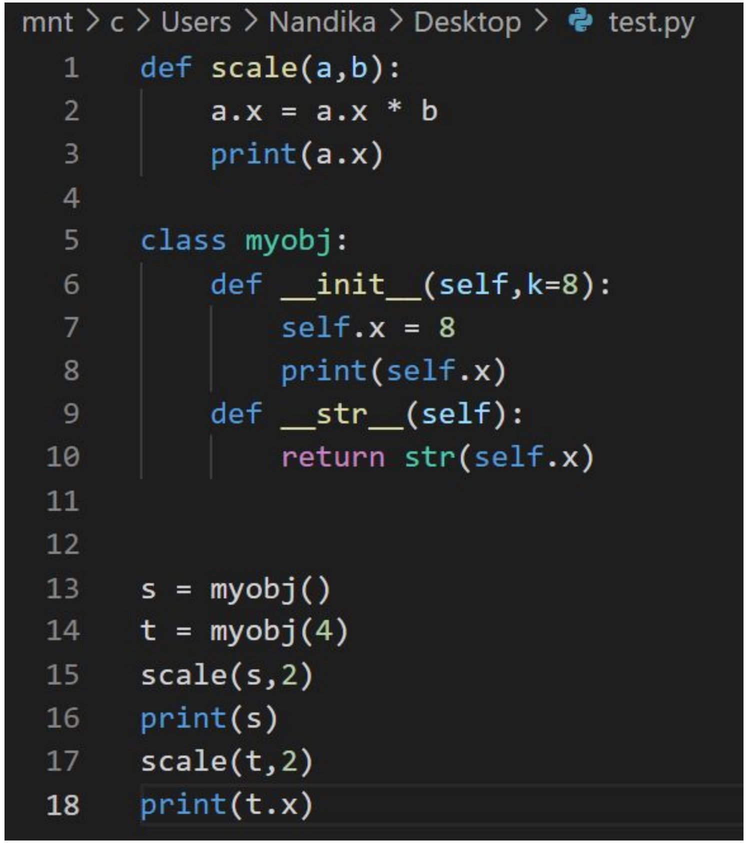
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Practice Question 4

Find the output of the following code:

__str__:

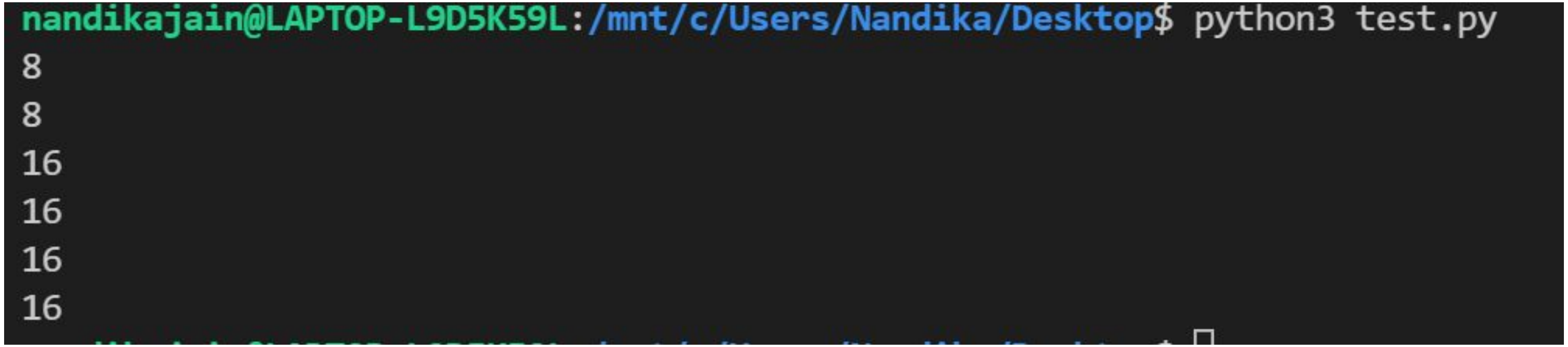
Python objects into strings.



mnt > c > Users > Nandika > Desktop > test.py

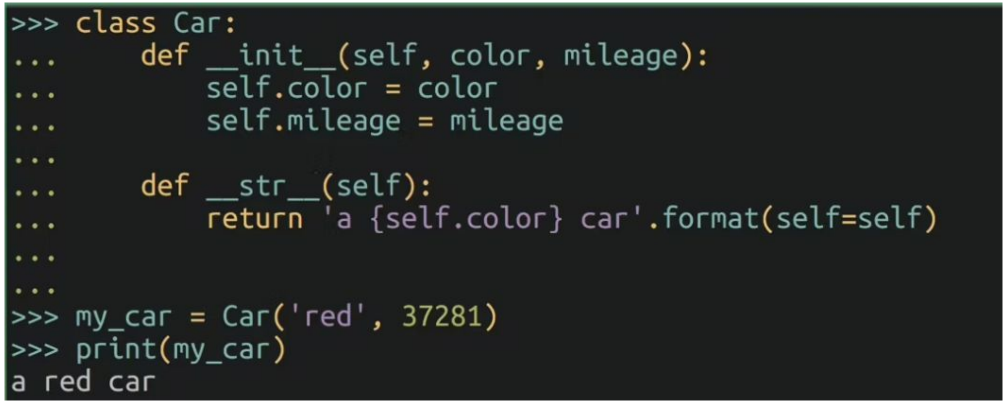
```python
def scale(a,b):
    a.x = a.x * b
    print(a.x)

class myobj:
    def __init__(self,k=8):
        self.x = 8
        print(self.x)
    def __str__(self):
        return str(self.x)


s = myobj()
t = myobj(4)
scale(s,2)
print(s)
scale(t,2)
print(t.x)
```

Ans to Practice Problem 4



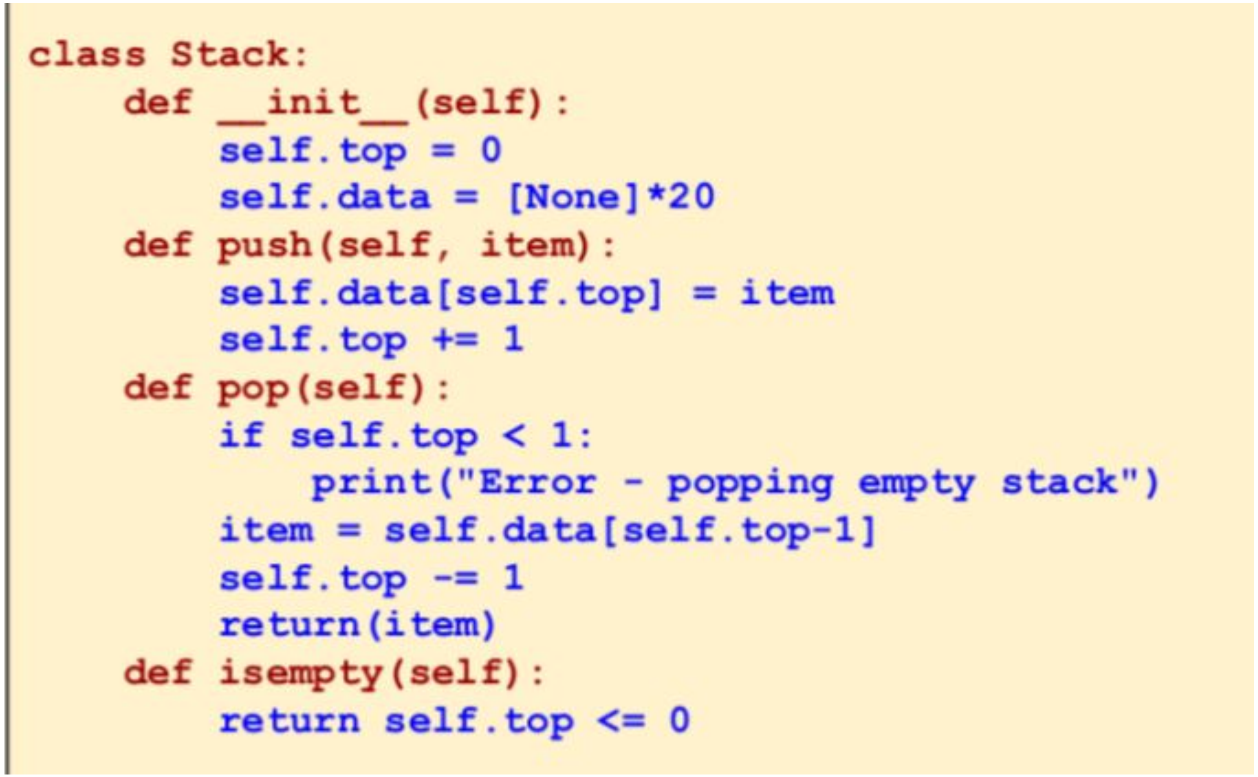
```
nandikajain@LAPTOP-L9D5K59L:/mnt/c/Users/Nandika/Desktop$ python3 test.py
8
8
16
16
16
16
```

__str__ Example:



```
>>> class Car:
...     def __init__(self, color, mileage):
...         self.color = color
...         self.mileage = mileage
...
...     def __str__(self):
...         return 'a {self.color} car'.format(self=self)
...
...
>>> my_car = Car('red', 37281)
>>> print(my_car)
a red car
```

Stack: LIFO



```
class Stack:
    def __init__(self):
        self.top = 0
        self.data = [None]*20
    def push(self, item):
        self.data[self.top] = item
        self.top += 1
    def pop(self):
        if self.top < 1:
            print("Error - popping empty stack")
        item = self.data[self.top-1]
        self.top -= 1
        return(item)
    def isempty(self):
        return self.top <= 0
```

What will be the changes if we have to use the same code to implement queue?

Postfix

Let the given expression be “2 3 1 * + 9 -“. We scan all elements one by one.

1) Scan ‘2’, it’s a number, so push it to stack. Stack contains ‘2’

2) Scan ‘3’, again a number, push it to stack, stack now contains ‘2 3’ (from bottom to top)

3) Scan ‘1’, again a number, push it to stack, stack now contains ‘2 3 1’

4) Scan ‘*’, it’s an operator, pop two operands from stack, apply the * operator on operands, we get 3*1 which results in 3. We push the result ‘3’ to stack. The stack now becomes ‘2 3’.

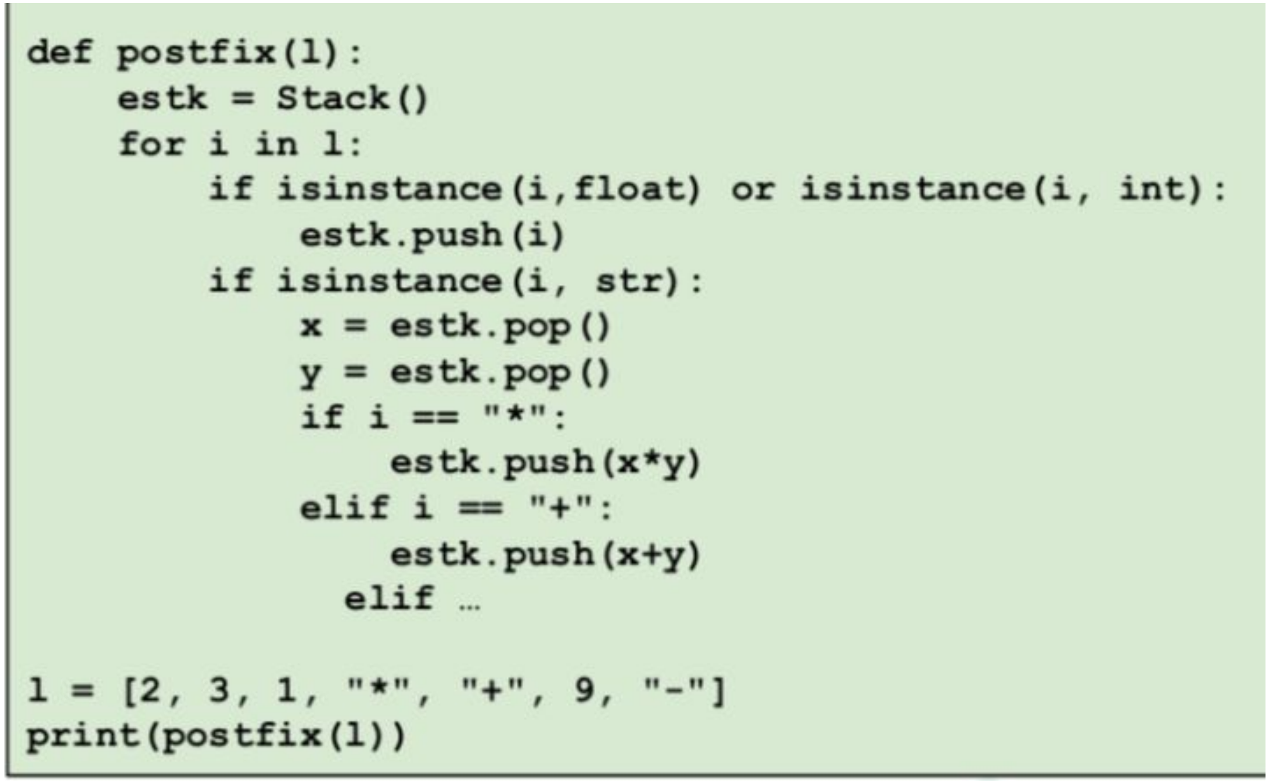
5) Scan ‘+’, it’s an operator, pop two operands from stack, apply the + operator on operands, we get 3 + 2 which results in 5. We push the result ‘5’ to stack. The stack now becomes ‘5’.

6) Scan ‘9’, it’s a number, we push it to the stack. The stack now becomes ‘5 9’.

7) Scan ‘-‘, it’s an operator, pop two operands from stack, apply the – operator on operands, we get 5 – 9 which results in -4. We push the result ‘-4’ to the stack. The stack now becomes ‘-4’.

8) There are no more elements to scan, we return the top element from the stack (which is the only element left in a stack).

Postfix Expression Evaluation using Stack

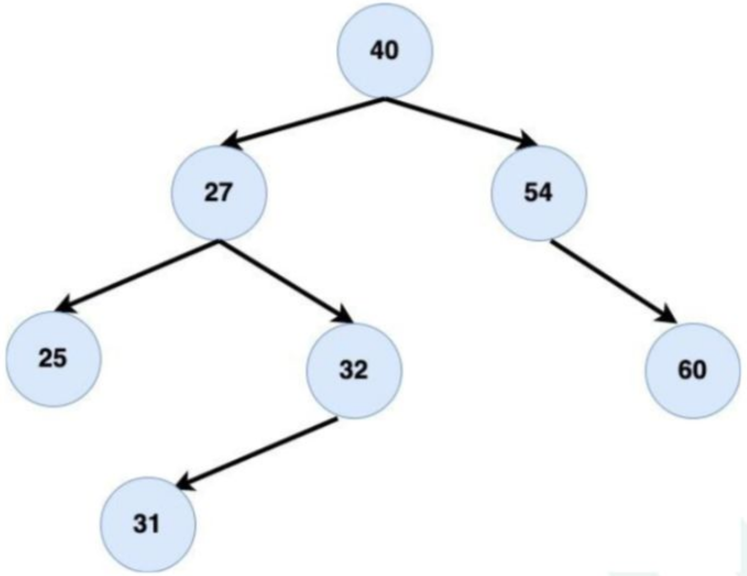


```
def postfix(l):
    estk = Stack()
    for i in l:
        if isinstance(i,float) or isinstance(i, int):
            estk.push(i)
        if isinstance(i, str):
            x = estk.pop()
            y = estk.pop()
            if i == "*":
                estk.push(x*y)
            elif i == "+":
                estk.push(x+y)
              elif …

l = [2, 3, 1, "*", "+", 9, "-"]
print(postfix(l))
```

BST

Example – Binary Search Tree

- Binary search tree - a root with value, and links to a tree on left and right
- Left sub-tree has values smaller than the root
- Right subtree has values greater than the root
- Searching for an element with binary search tree is very efficient - recursion is natural here
- Printing sorted items is easy



BST

```python
class Tree:
    #Attributes: data, ltree, rtree
    def __init__(self, val):
        self.data = val
        self.ltree = None
        self.rtree = None

    def insert(self, val):
        if val == self.data:
            return
        if val < self.data:
            if self.ltree is None:
                self.ltree = Tree(val)
            else:
                self.ltree.insert(val)
        elif val > self.data:
            if self.rtree is None:
                self.rtree = Tree(val)
            else:
                self.rtree.insert(val)
```

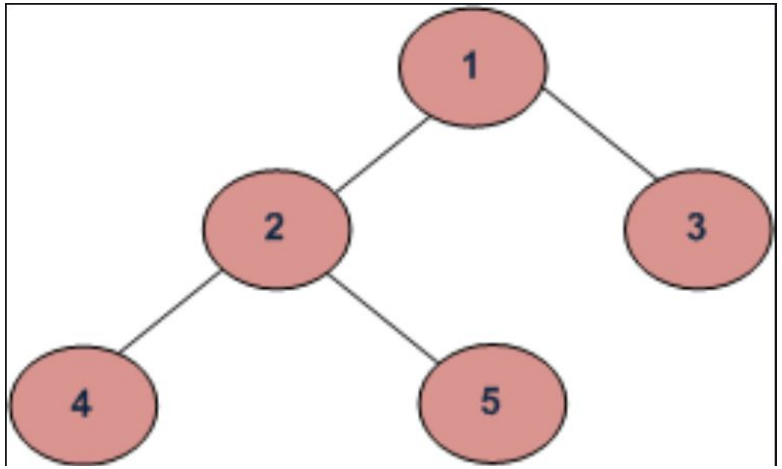
```python
    def ispresent (self, val):
        if val == self.data:
            return  True
        if val < self.data:
            print("Search L subtree")
            if self.ltree is None:
                return False
            else:
           return self.ltree.ispresent(val)
        elif val > self.data:
            print("Search R subtree")
            if self.rtree is None:
                return False
            else:
           return self.rtree.ispresent(val)
```

Tree Traversals

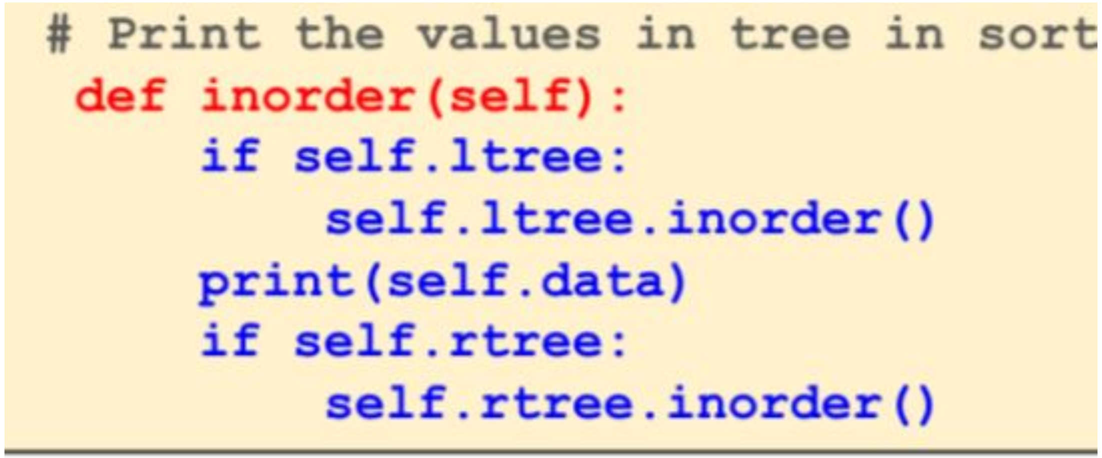
(a) Inorder (Left, Root, Right) : 4 2 5 1 3

(b) Preorder (Root, Left, Right) : 1 2 4 5 3

(c) Postorder (Left, Right, Root) : 4 5 2 3 1



Code for Tree Traversal



```
 # Print the values in tree in sort
  def inorder(self):
      if self.ltree:
          self.ltree.inorder()
      print(self.data)
      if self.rtree:
          self.rtree.inorder()
```

Inorder: Returns in sorted order

What will be the changes for preorder and postorder??