Object Oriented Programming

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Recap

- Class allows programmer to define new user-defined types - with operations on the the type and attributes to perform them
- Objects can be created of this type - and operations defined in the class can be performed on these objects
- We have seen some types - Complex, Stack, Tree
- Using classes effectively and to use OO concepts in programming requires practice and use
- Objects of user defined classes are mutable - like lists; so assigniing o2 = o1, only passes the pointer in o1 to o2 - both pointing to same object

Example

- Let’s take another common abstract type - wait queue
- It is used in a host of applications - to maintain a queue of people waiting to be served (at a doctor, hospital, test, some restaurant, office hour of a faculty..)
- Let us create a type for waiting queue, which we can then use for creating the waiting queue
- Let us first see a use case of this queue - and from that identify the set of operations (methods) needed; then build the class

A use-case for the queue

- Suppose we want to build a program to manage wait queue for a prof's office hour - in this application
  - A student comes, the assistant adds him/her to the queue
  - When prof is free, the asst removes the first student in the queue, prints info about the student; if queue is empty, then lets the prof know
  - If Prof wants to know many students are still waiting - give the number
  - When the office hour ends, gives a list of students who could not be served
  - …
- We will develop a Queue of persons to manage the wait queue of students for the prof
  - This Queue type can be used in other such applications also

Methods/Operations on Queue

- What operations do we want in this queue in which we will maintain the people (students) waiting
  - add(person): when a new person comes - he/she is added
  - remove(): returns the person who is to be now helped
  - isempty(): is the queue empty
- We also need methods to
  - Give the length of the queue: how many people are in the queue
  - Print the list of people waiting in the queue
- Let us now define a class Queue to implement this type - we will first implement the first few operations

Class Queue

```
class Queue:
    def __init__(self):
        self.qdata = []
        self.front = 0
        self.end = 0

    def add(self, obj):
        self.qdata.append(obj)
        self.end += 1
```

```
def remove(self):
        if self.isempty():
            return None
        else:
            obj = self.qdata[self.front]
            self.front += 1
            return obj

def isempty(self):
      if self.front == self.end:
            return True
      else:
            return False
```

Using the Queue – office hour

```
def studentq():
    waitroom = Queue()
    while True:
        op = input("1: add, 2: remove, -1: exit: ")
        op = int(op)
        if op == 1:
            rollno = input("Give roll no: ")
            waitroom.add(rollno)
        elif op == 2:
            obj = waitroom.remove()
            if obj == None:
                print("No student waiting")
            else:
                 print(f'Next student: {rollno}')
        elif op == -1:
            break
        else:
             print("Incorrect command, try again")
```

- The program / appl is used by the prof's asst - who gives commands
- This appl not need to know internals of how the Queue is implemented
  - Just like you don't need to know how list, dir … are implemented
- Code for class Queue can be written by one programmer and provided to another, who can use it by importing it as a module
  - We have put the user code in the same file for simplicity. We will see its use as a module

Dunder methods – impl. some std fns work

- With objects of list, dir, set, … you can perform some std ops
- E.g. print () for a list, dir… prints the object
- E.g. calling len() on list, dir, set .. gives you the number of items
- If these were available for Queue, we can use them to provide the # waiting, or students in the queue
- The semantics of each of these depends on the type - so python only defines it for the types it has

- How can we provide these for user defined classes?
- Dunder methods provide this - these are methods whose name starts with "__" (**d**ouble **under**score - hence the name)
- Dunder methods cannot be directly called on objects

The __str__ method

- How do we print an object - for lists, sets, etc - the print() prints it in some format (we can also do str(list/set/..) to get a string equivalent)
- For a class object o, if you print(o), you will get the type of the object and a memory location where it is. For example,

```
#<__main__.Complex object at 0x7fbfb42ae4c0>
```

- To print an object we can provide suitable method to print relevant info - and then invoke it on the object
- Or we can provide a method __str__(self) to return a string - containing info about the state of the object. With __str__(), when print(o) is called, __str__() is invoked, which returns a string that gets printed
- __str__() is also called when str(obj) is called

Example: __str__

```
class Complex:

   def __init__(self, r, im):
     self.real = r
     self.img = im


   def __str__(self):
           return  f'r:{self.real}  +
i:{self.img}'
```

```
# in Queue class with front, end

def __str__(self):
        s = ""
        for i in range(self.front,
self.end):
            s = s+str(self.qdata[i])+"; "
        return s


# Note: for data on the object in the queue
it calls str on that object (which will use
__str__ defined on that class)
```

The __len__ method

- If we call len(obj) on a class object: TypeError comes
- If we have a __len()__ method in a class, then that method is called by the function len(obj)
  - This is what python does for list, dir, etc - their impl have this method
- Instead of writing a special function for length for a class (e.g. qlen()) , it is better to define this dunder method and use len()

```
def __len__(self):
        return self.end - self.front
```

- There are other functions also for which dunder methods can be defined - we will not discuss any more

Some std fns that work on Class Objects

- There is a function *is* - which takes two vars and returns True if both vars point to the same object
  - x is y # returns True if x is same object as y, i.e. both point to same obj
  - This will work for user defined classes also - as it just needs to check the value of the two vars
- There is another function isinstance(*obj*, *type*) - this returns True if the object *obj* is of the type *type*
  - This will also work on class type objects
- These operations works for user defined class objects also

Quiz : Single Correct

Q) What would be the output of the following code snippet:

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def incrsalary(self, n):
        self.salary += n
        return salary

emp1 = Employee("John",10000)
emp2 = emp1
sal = emp1.incrsalary(10000)

print(emp1 is emp2)
print(sal)
```

A) True
   10000
B) False
   10000
C) True
   20000
D) False
   20000

Quiz : Answer

Q) What would be the output of the following code snippet:

A) True
10000

B) False
10000

C) True
20000

D) False
20000

Explanation : emp1 and emp2 point to the same object.

```
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
    def incrsalary(self, n):
        self.salary += n
        return salary

emp1 = Employee("John",10000)
emp2 = emp1
sal = emp1.incrsalary(10000)

print(emp1 is emp2)
print(sal)
```

Using Queue for different object types

- We have seen code for students waiting - using Queue
- We can use Queue for a waiting queue of patients in a hospital
  - The Queue class can be used - objects of Patient types will be passed
- In Queue, type of obj not imp for add/remove/len..; but both object types have different attributes when info about object is to be shown, Queue has to be coded smartly
  - To show information about the object - the queue method calls the str function on the object, which calls __str__
- With this Queue class can be used to create queues for different types of objects
- Methods like __str__, __len__ etc facilitate development of general modules which can be used in multiple applications

Using Queue for Students

```
class Student:
    def __init__(self, name, rollno):
        self.name = name
        self.rollno = rollno
        self.gradyr = None
    def setgradyr(self, yr):
        self.gradyr = yr
    def __str__(self):
        return f'{self.name},
{self.rollno}, {self.gradyr}'
```

```
from myqueue import Queue

def studentq():
    qs = Queue()
    s1 = Student("one", 11)
    qs.add(s1)
    s2 = Student("two", 22)
    qs.add(s2)
    s3 = Student("three", 33)
    s3.setgradyr(2022)
    qs.add(s3)
    print("Student queue:", len(qs))
    print("Students:", qs)
```

Using Queue for Patients

```
class Patient:
    def __init__(self, name, mobile, age, sex):
        self.name = name
        self.mobile = mobile
        self.age = age
        self.sex = sex
        self.disease = None
    def __str__(self):
        return f'{self.name, self.age}'
    def diagnosis(self, dname):
        self.disease = dname
```

```
def patientq():
    pq = Queue()
    p1 = Patient("ek", 123, 25, "M")
    p2 = Patient("do", 156, 36, "F")
    pq.add(p1)
    pq.add(p2)
    print("Patient queue: ", len(pq))
    print("Patient:", pq)
```

Object Comparisons

- For list, strings, .. you are often provided operations like +, *, -..
  - The result of these ops is also defined
  - E.g. + adds two lists, * repeats the list
- You can also check if two objects are equal by == (or <=, …)
- The operation == checks if two objects are equal (if they point to same object (i.e. is is True), then clearly == will be True)
- But if they are not the same object, then equality has to be defined
- For lists, sets, strings .. defined by python
- For class, we can use these operations by suitably defining some dunder methods
- E.g. for ==, a method __eq__() needs to be defined
  - If you check == on objects, without eq() definition, python converts it to checking for "is" (i.e. same)
- Similarly for other operations
- With these dunder methods defined, we can use the operations on objects

Quiz

Given the following class definition:

```
class MyObject:
    def __init__(self, value):
        self.value = value
    def __eq__(self, other):
        return self.value == other.value
```

What is the result of the following comparisons:

1) MyObject(10) == MyObject(10)

2) MyObject(10) is MyObject(10)

a) True, True

b) True, False

c) False, True

d) False, False

Quiz(Solution)

Given the following class definition:

```
class MyObject:
    def __init__(self, value):
        self.value = value
    def __eq__(self, other):
        return self.value == other.value
```

What is the result of the following comparisons:

1) MyObject(10) == MyObject(10)

2) MyObject(10) is MyObject(10)

a) True, True

b) True, False

c) False, True

d) False, False

Explanation: “==” will compare two objects based on the __eq__ dunder method, while “is” will check whether both points to same object or not

Dunder methods for some common ops

Operation	Dunder method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)
>=	object.__ge__(self, other)

Example – Queue class ops

```
# For == operation
def __eq__(self, q2):
        if len(self) != len(q2):
            return False
        for i in range(len(self)):
            if self.qdata[i] != q2.qdata[q2.front+i]:
                return False
        return True
```

```
# for + operation
def __add__(self, q2):
        for i in range(q2.front, q2.end):
            self.add(q2.qdata[i])
```

Copying objects

- Class objects are mutable (by defn) - their states can be changed
- So, obj1 = obj2, only provides another pointer to obj
- For list we have lst.copy() method provided by python
- What about copying objects of classes? Can write a copy() method
- Better - use the copy module provided by python

```
Import copy

q3 = copy.copy(q1)  # copies the queue q1 to q3
```

- copy() does a shallow copy - copies only objects, but not nested objects - so they may be pointers
- copy.deepcopy() - copies recursively if nested objects

Quiz – Single correct

Which of the following is the output of the code?

A. 15
15

B. 15
16

C. 16
15

D. 16
16

```
class Add:
    def __init__(self, a, b, c):
        self.sum = a + b + c

    def __add__(self, b):
        return self.sum + b.sum

    def __str__(self):
        return str(self.sum + 1)

obj_1 = Add(1,2,3)
obj_2 = Add(2,3,4)
obj_1.sum = obj_1 + obj_2
print(obj_1.sum)
print(obj_1)
```

Quiz – Single correct

Which of the following is the output of the code?

A. 15
15

B. 15
16

C. 16
15

D. 16
16

```
class Add:
    def __init__(self, a, b, c):
        self.sum = a + b + c

    def __add__(self, b):
        return self.sum + b.sum

    def __str__(self):
        return str(self.sum + 1)

obj_1 = Add(1,2,3) # sets obj_1 as 6
obj_2 = Add(2,3,4) # sets obj_2 as 9
obj_1.sum = obj_1 + obj_2 # 15 put in obj_1.sum
print(obj_1.sum) # prints attribute; 15 (6+9)
print(obj_1) # prints __str__ function; 15+1
```

Another Example – Triangle

- In geometry, we can specify a triangle by giving 3 points, each point being a tuple (x,y) coordinates
  - We want to find properties of this triangle - perimeter, area, is it isosceles or right or equilateral, …
- We can define a class for this triangle, and have methods to determine the perimeter, area, type, …
  - If coordinates of points are not important, can define class with attributes as three line lengths (determined by the three points given to init)
  - If coordinates are important, then coordinates of three points are attributes and we can define a function to determine line length

Triangle (lines as attributes)

```
import math
class Triangle:
    def __init__(self, p1, p2, p3):
        if p1==p2 or p2 == p3 or p1 == p3:
            print ("Not a Triangle: Two or more points are same")
            return
        self.l1 = math.sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)
        self.l2 = math.sqrt((p1[0]-p3[0])**2 + (p1[1]-p3[1])**2)
        self.l3 = math.sqrt((p3[0]-p2[0])**2 + (p3[1]-p2[1])**2)
    def equilateral(self):
        return self.l1 == self.l2 and self.l2 == self.l3 and self.l1 == self.l3
    def perimeter(self):
        return self.l1+self.l2+self.l3
    def area(self):
        s = (self.l1+self.l2+self.l3)/2
        tmp = s*(s-self.l1)*(s-self.l2)*(s-self.l3)
        return math.sqrt(tmp)
```

```
t0 = Triangle ((1,1), (2,3), (1,1))
t1 = Triangle ((1,1), (1,3), (4, 5))
print("Perimeter: ", t1.perimeter())
print("Area: ", t1.area())
```

Triangle …

- In this class, the attributes are l1, l2, l3 - the coordinates of the three points are passed as params to init - local vars
- As only line lengths are maintained, only some type of operations can be performed - type of triangle, area, perimeter, …
- Cannot perform other types of operations, e.g. rotate a triangle, move it in the 2-D plane, …
- Can also implement it by keeping the coordinates, and then computing the lines whenever needed
    - Note that to call a method from within a method, the object is referred to as "self" (which is the variable holding the pointer passed)

Triangle (points as attributes)

```
import math
class Triangle:
  def __init__(self, a,b,c):
    self.a = a
    self.b = b
    self.c = c
  # Compute distance between two points
  def distance(self, p, q):
    temp1 = (p[0]-q[0])**2
    temp2 = (p[1]-q[1])**2
    d = math.sqrt(temp1+temp2)
    return d
 # Compute the perimeter of the triangle
  def perimeter(self):
    ab = self.distance(self.a, self.b)
    bc = self.distance(self.b, self.c)
    ac = self.distance(self.a, self.c)
    return ab+bc+ac
```

```
 #Area computation by the Heron's formulae
  def area(self):
    ab = self.distance(self.a, self.b)
    bc = self.distance(self.b, self.c)
    ac = self.distance(self.a, self.c)
    s = (ab+bc+ac)/2
    temp = s*(s-ab)*(s-bc)*(s-ac)
    res = math.sqrt(temp)
    return res
```

```
t = Triangle((1,2),(3,-4),(-4,5))
print(t.area())
# 11.999999999999986
print(t.perimeter())
# 23.557261466173436
```

Triangle

- In this implementation, distance() is an internal method - used for providing the external facing methods - area(), perimeter()
- It should not really be invoked from outside
  - Note the args and parameters
- Ideally such methods should be "private" to the class and not visible outside
  - Some languages provide such mechanisms - some of the methods are declared as private and cannot be invoked from outside
  - However, python does not have a clean way (there is a way - we may discuss it later)

Summary – Classes and Objects

- Classes provide a way to define new data types
- Provides another abstraction / construct to write modular code
- For smaller problems / code in python - functions suffice; for large problems, classes and objects are common
- Basic OOP approach - define classes suitable for the problem, define methods on those, develop them separately (maybe as modules), then use them for problem solving by just using the operations on objects (and not worrying about internals)
- To use common functions (like print, len, …) and common relational operations (like ==, !=), dunder methods are useful

Minute Paper

Pls reflect and take a minute to fill the MP on basic class and objects- it helps your learning, and gives me feedback