Object Oriented Programming

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Python so far…

- We have seen different statement types - assignment, conditional, loops, …; can be used to write program logic
- We have seen one concept - functions - for dividing a program into logical parts, with each part doing some well defined computation or providing defined interface and functionality
  - This is called modularity - making a program modular
  - Such mechanisms for modularity/divide-and-conquer are essential for solving complex problems
  - Cannot write a program of thousands of lines as just one program - must break it into parts and develop parts separately
  - 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

Python so far...

- We have seen some basic data types (int, float, bool) - they provide the basic objects on which computation is done
  - 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
- Python has some built-in complex data types - lists, sets, dictionaries - each has its attributes / internal data, and a set of operations are defined
  - We can define variables of this type, and perform the operations on them
- What if we want to build a new data type - e.g. a queue, a graph, an n-dimensional vector, with specific operations on them?
- Object oriented constructs provide this capability - define new user defined data types

A Data Type

- A data type defines a collection of data objects and a set of predefined operations on those objects, e.g. types
  - int: objects of int with ops: add, subtract, divide, exponentiate, …
  - string: string objects with operations like: join, substring, concatenate, etc. (but not divide, exponent, …)
  - set: set objects with ops: check membership, union, intersection, etc
- To support the operations, whatever information needed is maintained by implementation of the type
  - E.g. sets are implemented as hash tables, has elements, …
  - Details of implementation hidden - user only uses the operations
- Programming languages provide a few predefined types
  - It will be nice if the language provides ability to define a new type - then we can define types needed for the problem we are solving
  - Have to define the operations on this type and implement them
  - Then in programs, can declare variables of this type, on which the defined operations can be performed, implementation of ops is hidden from user

Object Oriented Programming

- User defined data types capability in a prog. language
  - Ability to define a new type: the structure of an object of this type i.e. what attributes/vars it has, and the operations defined on objects of this type
  - Ability to define variables of this type, and invoke operations on it
- With such capability, we can have different user defined data types
  - Can define a "robot" type, which will have operations like move, lift, put, bend, etc and will have all data needed to perform these operations (e.g current location, status of arm, …)
  - 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, …
- With capability to define new types - we can view programming as working with objects of different types and doing operations on them - this is the OO view
- Procedural programming - program is for computation, and computation can be bundled as functions

Examples of User Types

- Stack: Last in first out - very useful in many applications
  - Ops: push, pop, isempty
- Queues: First in first out - most queuing systems use it
  - Ops: add, remove, isempty
- Binary search tree - a root with a left sub tree which has smaller values, and a right subtree with larger values
  - ops: add elt, remove element, search for an element, join trees, …
- Graph is a common way to represent many problems - e.g. cities with roads between them, any network (friendship, computer, …)..
  - ops desired: add a node, delete a node, add a link, delete a link, find path from A to B, find shortest path (if edges have values), check if a node exists, find all nodes connected to a node, …
- Complex numbers - we will use this as an initial example
  - Ops: add, subtract, findreal, findimag, …

Examples…

- In some problems, we might want to model a real world data object, for example a car along with its associated details and functions.
- Students in an Institute - we can create a dictionary with all values and then write functions on it; alternatively, we can define a Type student and then define as many students we want
  - Ops: What is the name, SGPA (sem), CGPA, graduated, list of courses, …
- While implementing an e-commerce site, may want to provide abstractions for customer, catalog, shopping_cart, …
- In a computer game, we can have characters, guns of different types, targets of different types, ….

Defining a Class (i.e. a new Type)

- Python allows defining a new type as a class
- To define a type for complex numbers:

```
class Complex:
    <body of the class definition>
```

- Declares that Complex is a type - like list, dict, set, int, ..
- As a type, objects of this type can be created and reference to them assigned to variables
- Generally, class Names start with a capital letter

Objects of a Class

- Class defines a new type (like list, set, etc), with a set of valid operations, and some attributes on which ops work
- We can create objects of this type and assign them to variables - as with language defined types like dict, sets, lists, int, ..
- An object of a class can be created and assigned to a var by

```
c1 = Complex()
```

- An object of type Complex is created and the var c1 points to that object
- Role of variable is same as with all types; the nature of object is now of Complex type

Class Methods and 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
  - Complex will have methods like: add, subtract, getreal, getimag, …
- A class has attributes - these are variables in its scope, accessed from within the class to implement the methods of the class
  - E.g. Complex has attributes real and imag
  - These attributes define the state of an object of this class
- As Python does not have explicit var declaration, attributes are defined in methods in the class, usually the __init__

The __init__ method

- Object can be created/instantiated without any parms
  - This creates an empty object (like an empty set, list, dict..)
  - Then will need methods to define the state of the object, i.e. after creation have some methods like setstate(args) to set the values
- Often one would want to give initial values to define the initial state of the object while creating it - Python provides a method
- If there is a method __init__ defined in the class, then on creation that method is called
- __init__ typically will get some values as params and use them to define the state of the object being created, i.e. assign values to attributes for the object
- If no __init__ provided, creation of an empty object (whose state can be defined with state by calling methods)

Quiz – Single correct

What is the correct way to create an object of Employee class?

```
class Employee:
    def __init__(self, id, name):
        self.id = id
        self.name = name
        self.company = "Nike"
```

A. `tom = new Employee(108, "Tom")`

B. `tom = Employee.__init__(108, "Tom")`

C. `tom = Employee(108, "Tom")`

D. `tom = Employee(108, "Tom", "Nike")`

Quiz – Single correct

What is the correct way to instantiate this Employee class in Python?

```
class Employee:
    def __init__(self, id, name):
        self.id = id
        self.name = name
        self.company = "Nike"
```

A. `tom = new Employee(108, "Tom")`

B. `tom = Employee.__init__(108, "Tom")`

C. `tom = Employee(108, "Tom")`

D. `tom = Employee(108, "Tom", "Nike")`

Class and Objects – calling Methods

- A class has no state, it is just a definition - the state is within objects, just like list itself has no state - only a list object we create has state
- An object can be created and initialized through the method __init__()
- On an object of a class, the methods that are defined in the class are the set of operations that can be performed - like on sets we can perform the operations defined on sets
- Operations defined in a class can be invoked on an object using

  object_variable.operation (args)

- 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

Method Calls and self

- There can be many objects of a class - each has its own state
- To execute the methods defined in class, we need to identify which object to perform them on
- When a method is invoked on an object, the reference to the object is passed implicitly to the method as an argument
- In the method, it is the first parameter **self** - the ref to the object
- All attributes of an object are accessed as self.attribute
- In methods, those variables associated with self define the class attributes - other variables defined method are local
- All objects have these attributes - each attribute has its own state
- In a method on an object - must access attributes through "self"

Example: Complex

```
class Complex:
    def __init__(self, r, i):
        self.real = r
        self.imag = i

    def getreal(self):
        return self.real

    def getimag(self):
        return self.imag

    def add(self, c):
        self.real += c.getreal()
        self.imag += c.getimag()

    def equal(self, c2):
        return  self.real==c2.getreal()and self.imag==c2.getimag()
```

```
# Main program
c1, c2, c3 = Complex(1,2), Complex(3,4), Complex(3,4)
print(c1.equal(c2))
c1.add(c2)
print(c1.getreal(), c1.getimag())
```

Object, Attributes, Methods

- For a class, the namespace is all the attribute names and all the method names - all are treated as attributes of the class
- All attributes of an object can be accessed, eg:

  object.attribute

- Some languages allow access to attributes only from within the class definition, i.e. by methods
- For supporting data encapsulation, state of an object should not be directly used - only methods should be invoked
- We will in general only invoke methods on objects

Execution with Classes

- When a program is executed, the class definitions are noted by interpreter, but no method is looked at
- When an object is created, then the __init__() method is visited and executed
- 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
- You can see this in pythontutor

Quiz – Single correct

What is the output of this code?

A. 7

B. 8

C. 9

D. Error

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

    def get(self):
        return self.sum

    def set(self, v):
        self.sum = v


obj = Add(2,3,4)
val = obj.get()
obj.set(val-1)
print(obj.get())
```

Quiz – Single correct

What is the output of this code?

A. 7

B. 8

C. 9

D. Error

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

    def get(self):
        return self.sum

    def set(self, v):
        self.sum = v


obj = Add(2,3,4) # sets sum as 9
val = obj.get() # val is 9
obj.set(val-1) # sum set as 8
print(obj.get()) #returns 8
```

Exercise for students

- Take the code of Complex, and add new operations on complex, eg:
- sub (), modulus(), conjugate(), …
- In pythontutor - and see what happen

Recap – new types

- Python provides: Basic primitive types (int, float, boolean)and some compound/structured types (list, dict, sets)
  - Programs (statements in python) can create objects of this type and perform the language provided operations on them
- With object-orientation it provides the capability to define user-defined types and use them in our programs
  - Programs can create objects of these types and perform the defined operations on them
- Many problems require objects of different types - with OO we can define a type for these objects and then use it

Recap – Modularity and abstraction

- Earlier we have learned one method of modularizing a program - i.e. partitioning the solution (program) into different components/modules, each with a clear abstraction and interface, and so can be developed and tested independently, and then used in the program
  - The construct is functions - we can partition a program into functions, develop them separately, and then combine them (e.g. in main)
  - Functions provide abstraction of a function - takes data inputs, computes something, and returns some data
- OO provides another method for modular programming - we can now define classes and methods, which are cohesive modules with clear abstraction and operations
  - This is another abstraction approach - here detailed data is hidden
  - We only create objects of this type and perform operations on them

Recap…

- New types are created by defining a class:

```
class <class_name>:
    class_body
```

- A class body has
  - Methods which provide operations on objects of this type
  - Attributes which keep the state of an object and on which methods work
  - A method __init__(), which is invoked when an object is created - generally used to initialize the state of the object
- Objects of a class are created by:

  var_name = class_name (args)
- Operations on an object `obj` are invoked using dot (.) notation:

  obj.method_name(args)

  Object reference is passed implicitly - within a method, object ref is in **self** parameter

Recap – Information Hiding / Abstraction

- Class - encapsulates data and provides operations
- So data is hidden from users, only operations can be performed
- This is what is desired - users want to perform operations - manipulating data was done only to get the ops
- Using classes and object - another way of thinking on how to design programs - very useful abstraction

Quiz – multicorrect

Which of these are true

A. OO provides another way of modularizing code
B. OO encapsulates the data and provides operations on the data
C. Without OO we cannot write big programs
D. In a Class, the set of methods should together implement the desired operations on the Class type

Quiz – Answer

A, B, D

True:

- OO provides another way of modularizing code
- OO encapsulates data and provides operations on the data for a program to use
- In a Class, the set of methods should together implement the desired operations on the Class type

False

- Without OO we cannot write big programs

Example: Postfix Expression Evaluation using Stack

- Stack is a common type - used in many applications
- Arithmetic expressions can also be written in postfix (also called reverse polish) notation - no parenthesis is required
  - The operator is mentioned after the two operands
  - The result of the operation replaces the two operands and op

    `(a+b)*c => ab+c* ; a/b+c/d => ab/cd/+; (a+b)*(c+d) => ab+cd+*`
- Evaluating postfix notation is efficient - best way is to use a stack

  If operand -> push it on a stack

  If operator -> pop last two values, apply operator, push result on stack

  When expression ended -> the only value on stack is the answer
- We need to create a stack data type for this

Using stack for postfix

- Type Stack defined as class wt ops: init(), pop(), push(), isempty()
- To evaluate an expression
  - Create a stack first by __init__
  - Then scan the expression - if operand then push; if operator then pop two elts and apply operator, push the results on stack
  - When expression scanned, the value remaining is the value
- There are many uses of stack - execution of functions in programs need stack (as we see on pythontutor when invoking functions)
- Often there will be other operations on Stack - top(), height(), ..

Postfix eval using stack...

```
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
```

```
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))
```

Postfix eval – code

```
class Stack:
    def __init__(self):
        self.top = 0
        self.data = [None]*20
    def push(self, item):
        print("Pushing: ", 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
        print("Popping: ", item)
        return(item)
    def isempty(self):
        return self.top <= 0
    def topelt(self):
        return self.data[self.top-1]
```

```
def postfix(l):
    estk = Stack()
    for i in l:
        #print(i)
        if isinstance(i, float) or isinstance(i, int):
            estk.push(i)
        if isinstance(i, str):
            x = estk.pop()
            y = estk.pop()
            if i == "*":
                print(f'performing {x} * {y}')
                estk.push(x*y)
            elif i == "+":
                print(f'performing {x} + {y}')
                estk.push(x+y)
            elif i == "-":
                print(f'performing {x} - {y}')
                estk.push(x-y)
            else:
                print("Dont know the operation")
    x = estk.pop()
    return x
l = [2, 3, 1, "*", "+", 9, "-"]
print(postfix(l))
```

Postfix using Stack...

- Main external interface of a type is the operations (methods) - how it is implemented is hidden
  - Programmer for postfix() need not know anything about internals
  - It only needs to know the operations of stack
- We have implemented stack using lists - the most natural way; however we can implement it in other way (e.g. dictionary)
- With classes, if the implementation is changed - it does not affect the use/invocation as long as method interfaces remain the same
- This is a key property of OO (any abstraction) - internals can be changed later without impacting users of a class

Exercise (a few mts)

- For office hours of a faculty we want to build a waiting system where a student can come and join the waiting room, and as the faculty is free the earliest student who came will be called
- If no student in the waiting room, faculty is informed when "next" student is called (and when a student comes, he/she
- Write the structure of a class "Waitroom" for this
  - Q1: First what operations it should have and their interface
  - Q2: What attributes you will have for this class to impl
-

Student queue – code

```
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
```

```
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")
    print("office Hr ended")
    print(f'No waiting: {waitroom.end-waitroom.front}')
studentq()
```