Chapter: OO

What is an object

- Which occupies space
- Which may have data
- Which may have behavior

- For e.g. In real world “Car” is an object which has a “name” (data) and which can drive (behavior)

What is an Class

- Objects of similar type will have similar attributes and similar behaviour
- All objects of car have a name, can drive
- Or in other words if we somehow create a type then we can use that as a template/blueprint to create objects from that
- Such a template or blueprint is called as class

- For eg.
- a=10;b=20;c=30 are all “objects” of the “class” int

- For eg.
- We are all “objects” of “class” called “human beings”

Create a class in python

- A pretty useless car. No data No behaviour

```
>>> class Car():
        pass
>>> b = Car()
```

PYTHON

Create an object in python

- Oh a little better at least it has a name

```
>>> class Account():
        def __init__(self,holdername, acctype="Savings"):
                self.balance = 1000.00
                self.acctype=acctype
                self.holdername = holdername


>>> ac = Account('Aditya')
```

PYTHON

Pretty print an object

```
>>> class Account():
        def __init__(self,holdername, acctype="Savings"):
                self.minbalance = 1000.00
                self.acctype=acctype
                self.holdername = holdername
        def __repr__(self):
                return '{}, {}'.format(self.holdername, self.acctype)
        def __str__(self):
                return 'Account Holder : {}, Acc Type :
{}'.format(self.holdername, self.acctype)
```

Object of an account

- So now an Object of account has data (balance, holdername, acctype)
- So now an Object of account has behaviour (print)

- In other words we can now say that the object has encapsulated data and behaviour

Encapsulation

Add more behaviour

```
def credit(self, amount=0):
    self.balance += num
    return self.balance
def debit(self, amount=0):
    if(self.balance - amount < 0):
       # raise an exception
    else
       return self.balance -= amount
```

Raising an exception

```
def debit(self, amount=0):
    self.balance += num
    return self.balance
def credit(self, amount=0):
    if(self.balance - amount < 0):
       # raise an exception
    else
       return self.balance -= amount
```

PYTHON

Living Being – Human Being inherits

```
>>> class LivingBeing():
        def breathe(self):
                print "I can
breathe, therefore I am a
LivingBeing"
```

```
>>> class HumanBeing(LivingBeing):
        pass
```

Living Being – Human Being inherits

```
>>> class LivingBeing():
        def breathe(self):
                print "I can 
breathe, therefore I am a 
LivingBeing”
```

```
>>> class HumanBeing(LivingBeing):
        def breathe(self):
                print "breathe 
through lungs. O2 in CO2 out"
```

Equality?

```
>>> class Student():
        def __init__(self, stud_id, stud_name):
                self.stud_id = stud_id
                self.stud_name = stud_name


>>> b = Student(1, 'John Doe')
>>> c = Student(1, 'John Doe')
>>> b == c
False
```

PYTHON

Equality?

```
class Student():
       def __init__(self, stud_id, stud_name):
              self.stud_id = stud_id
              self.stud_name = stud_name
       def __eq__(self, stud2):
              return self.stud_id == stud2.stud_id

>>> b = Student(1, 'John Doe')
>>> c = Student(1, 'John Doe')
>>> b == c
True
```

Methods for comparision

```
__lt__(self, other)
__le__(self, other)
__eq__(self, other)
__ne__(self, other)
__gt__(self, other)
__ge__(self, other)
```

PYTHON

Methods for Math

```
__add__(self, other)
__sub__(self, other)
__mul__(self, other)
__mod__(self, other)
__pow__(self, other)
```

Other

```
__str__(self)
__repr__(self)
__len__(self)
```

PYTHON

List Sorting

```
>>> sorted(student_objects, key=lambda student: student.age)
```

PYTHON

Classes - Challenge

```
Create a class to represent complex numbers (for eg. 5 + i6)
c = Complex(5,7)
d = Complex(3,7)
c + d should give 8 + i14
```