28th-June (characteristics of OOPs are same in all lang. but implementation is diff.)

Python - tutorial 6

- Abstraction
  - Public : accessible anywhere
  - private : __variable inside class
  - Protected : _variable inside & outside class within package

In python we don't use keywords public, private or protected like other prog. lng., instead we use notations

* Protected variables:

(_before variable = protected) (not all variables necessarily be protected).

```
class Person:
    def __init__(self, name, surname, dob):
        self._name = name
        self._surname = surname
        self._dob = dob
    def age(self, current_year):
        return current_year - self.dob
    def __str__(self):
        return "%s %s was born in %d." \
               %(self._name, self._surname
                 , self._dob)
alec = Person("MADHU", "JAIN", 1999)
print(alec)
print(alec._surname)   (: obj._variable)
```

→ MADHU JAIN was born in 1999

→ JAIN

print(alec.surname)
↳ error (: 'Person' object has no attribute 'surname')

* Private variables

```
class Person:
    def __init__(self, name, surname, dob):
        self.__name = name
        self.__surname = surname
        self.__dob = dob
    def age(self, current_year):
        return current_year - self.__dob
    def __str__(self):
        return "%s %s was born in %d" \
            %(self.__name, self.__
              surname, self.__dob)
alec = Person("MADHU", "JAIN", 1999)
print(alec._Person__name)
```

(can be mixture of public, protected & private)

→ MADHU (obj._class__variable)

```
alec.__dict__
```

→ {'_Person__name': 'MADHU',
'_Person__surname': 'JAIN',
'_Person__dob': 1999}

* Abstracting functions:

```
class tasty:
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def test(self):
        return "This is Public method."
    def _test1(self):
        return "This is Protected method."
    def __test2(self):
        return "This is private method."

obj = tasty(3,4,5)
obj.test()
    This is Public method.
obj._test1()
    This is Protected method.
obj.__test2()  obj._tasty__test2()
    This is private method.
```

- Inheritance : (Parent & Child classes)

```
class xyz:                          → parent class
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def test(self):
        return "This is public."
obj = xyz(4,5,6)
obj.test()
```

This is public.

Child class ← `class abc(xyz):` → (class abc is inheriting all properties of class xyz)

```
    pass
obj1 = abc(6,7,8)
obj1.b
```

→ 7

(*) What if we wish to inherit only few properties ? (as per requirement) & also if we wish to create some new variables in child class ?

```
class abc(xyz):
    def __init__(self,a,b,m,n):
        xyz.__init__(self,a,b)
        self.m = m
        self.n = n
```

*args → all arguments
*kwargs → key-word arguments (in case of dictionary)

```
obj2 = abc("MA", "DU", 5, 6)
obj2.m → 5
```

(*)
```
class abc(xyz):
    def __init__(self,a,b,m,n):
        super().__init__(a,b)
        self.m = m
        self.n = n
obj2 = abc("MA","DU",5,6)
print(obj2.m) → 5
```

Check: isinstance(obj2, xyz) → True

(*) two class A & B are inherited by class C such that A & B have same definitions.

(Multiple):

```
class A():
    def test(self):
        print('A')
class B():
    def test(self:)
        print('B')
class C(A,B):
    objA = A()
    objB = B()
objA.test() → A
objB.test() → B
```

```
obj = C()
obj.test() → A
```

(∵ the class which is inherited first by child class will be given preference
eg: here, C(A,B))

if class C(B,A)
then, obj.test() → B

So, here, C(A,B) ≠ C(B,A)

(Multilevel):

```
class A():
    def test(self):
        print('This is A')
class B(A):
    def xyz(self):
        print('This is B')
class C(B):
    def pqr(self):
        print('This is C')
objc = C()
objc.test()  ----> This is A
```

- Overriding Methods: (not changing function name/signature just changing body).

```
class A:
    def test(self, a=5):
        print("This is class A and 'a' is", a)
class B:
    def test(self, a=10):
        print("This is class B and 'b' is", b)
```

Note: signature should be kept same while method overriding.

(overriding & overloading are different).