ED5340 - Data Science: Theory and Practise

L10 - Classes and Objects

Ramanathan Muthuganapathy (https://ed.iitm.ac.in/~raman)
Course web page: https://ed.iitm.ac.in/~raman/datascience.html
Moodle page: Available at https://courses.iitm.ac.in/

Classes and objects

A ‘rough’ idea of what they are

- Classes are similar to structures in C
- Objects are similar to variables that access the member of the structure.

Classes and objects

- Class contains data and methods that can access or manipulate this data.
- Data is typically accessed through the methods (data protection).
- The methods are accessed through ‘object’ instantiation.
- Broadly comes under object-oriented programming (others being functional, structural programming).

Class details

Name, data, methods (member functions) - L10_class_example.py

```
class StudentDetail:

    def datainput(self, n, r, s):
        self.name = n
        self.rollno = r
        self.sem = s

    def printout(self):
        print(self.name, self.rollno, self.sem)

s1 = StudentDetail( ) #Object instantiation
s1.datainput('Ram', 12, 3)
s1.printout( )
```

Class - Using constructor

Name, data, methods (member functions) - L10_class_example.py

```
class StudentDetail:

    def __init__(self, n='', r=1, s=1):
        self.name = n
        self.rollno = r
        self.sem = s

    def printout(self):
        print(self.name, self.rollno, self.sem)

s2 = StudentDetail('Ram', 12, 3 ) #Object instantiation
s2.printout( )
```

Public Data

Name, data, methods (member functions)

```
class StudentDetail:

    def datainput(self, n, r, s):
        self.name = n
        self.rollno = r
        self.sem = s

    def printout(self):
        print(self.name, self.rollno, self.sem)

s1 = StudentDetail( ) #Object instantiation
s1.datainput('Ram', 12, 3)
s1.printout( )
print('name = ', s1.name, 'rollno = ', s1.rollno, 'sem = ', s1.sem)
```

Private Data

Name, data, methods (member functions) - L10_class_example.py

```
class StudentDetail:

    def datainput1(self, n, r, s):
        self._name1 = n
        self._rollno1 = r
        self._sem1 = s

    def printout(self):
        print(self.name, self.rollno, self.sem)

s2 = StudentDetail( ) #Object instantiation
s2.datainput1('Shyam', 23, 34)
s2.printout( )
print('name = ', s2._name1)
```

Constructor

L10_class_constructor.py

```
class StudentDetail:

   #Constructor
   def __init__(self, n='R', r=1, s=1):
      self._name = n
      self._rollno = r
      self._sem = s

    #Printing the data
       def printout(self):
          print('name = ', self._name, ", " , 'roll no = ',  self._rollno, ", " , 'sem = ', self._sem)

       #destructor
       def  __del__(self):
          print('Del obj' + str(self))

    s1 = StudentDetail()
    s1.printout()

    s1 = StudentDetail('Ram')
    s1.printout()

    s1 = StudentDetail('Raman', 23)
    s1.printout()

    s1 = StudentDetail('Ramana', 23, 5)
    s1.printout()
```

Class variables and methods

L10_cmv.py

- One variable shared across all objects
- ‘self’ should not be used
- syntax: classname.variable
- similar rules for class methods (classname.method( ))
- similar to static members in C++

Notation - Convention

L10_pvt_example.py

- Class name starts with Caps
- single _ for notionally private variable
- __ (dunderscore) for strictly private
- __ used in data as well as methods (e.g.?)

Operator overloading

L10_class_complex.py

- a + b already defined
- Operator overloading is done for user defined classes, for e.g. class Complex.
- def add_comp(self, other):
    - c1.add_comp(c2)
- def __add__(self, other)
    - c1 + c2   #(More intuitive usage)

Operators that can be overloaded

- __sub__
- __mul__
- ………… (find out the list of operators that can be overloaded)

CW: Do the + and - for the Complex class.

Dynamic creation of attributes

L10_dy_creation.py

```
class Passbook:

   pass

p1 = Passbook( )

p1._name = ‘Raman’

p1._number = 1234
```