



Object Oriented Programming using Python (I)

Lecture(4)

Creating class and object

Prepared by: **Dr. Rula Amjed Hamid**
University of Information Technology and Communications
College of Business Informatics

Instances of Classes

Abstract
concept



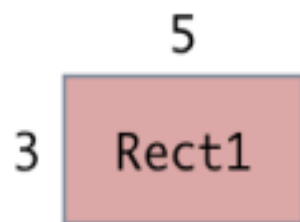
Attributes

- width
- height
- colour

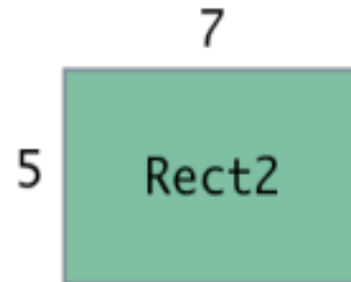
Functions

- area()

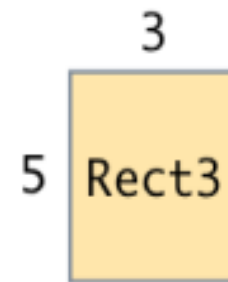
Multiple
concrete
instances



width=5
height=3
colour=(200,0,0)
area() -> 15



width=7
height=5
colour=(0,200,0)
area() -> 35



width=3
height=5
colour=(255,200,0)
area() -> 15

Creating Classes

Defining a class in Python is done using the class keyword, followed by an indented block with the class contents.

Class data
attributes



```
class <Classname>:  
    data1 = value1  
    ...  
    dataM = valueM  
  
    def <function1>(self, arg1, ..., argK):  
        <block>  
    ...  
    def <functionN>(self, arg1, ..., argK):  
        <block>
```

Class
member
functions



Defining Functions in Classes

- A class definition block can include multiple functions.
- These represent the functionality or behaviors that are associated with the class.

```
>>> class Maths:  
...     def subtract(self,i,j):  
...         return i-j  
...  
...     def add(self,x,y):  
...         return x+y
```

Argument (self) refers to the object itself

Calling Functions in Classes

- Using Class Functions from Outside a Class
Functions are referenced by using the dot syntax:
<objectName>.<methodName>()

```
>>> m = Maths()  
>>> m.subtract(10,5)  
5  
>>> m.add(6,7)  
13
```

← No need to
specify value for
self, Python does
this automatically

```
class Maths:  
    def subtract(self,i,j):  
        return i-j  
  
    def add(self,x,y):  
        return x+y
```


Calling Functions in Classes

- Using Class Functions from Inside a Class

When referring to functions from within a class, we must always prefix the function name with self (e.g. self.subtract())

```
>>> class Maths:
...     def subtract(self,i,j):
...         return i-j
...
...     def testsub(self):
...         print self.subtract(8,4)
```

Tell Python to use function
associated with this object



Attributes

Class attribute
defined at top of
class

```
>>> class Person:
...     company = "ucd"
...
...     def __init__(self):
...         self.age = 23
```

Instance attribute
defined inside a class
function.
The **self** prefix is
always required.

```
>>> p1 = Person()
>>> p2 = Person()
>>> p1.age = 35
>>> print p2.age
23
```

Change to instance attribute **age**
affects only the associated
instance (p2)

```
>>> p1 = Person()
>>> p2 = Person()
>>> p1.company = "ibm"
>>> print p2.company
'ibm'
```


Change to class attribute **company**
affects all instances (p1 and p2)

Constructor

- When an instance of a class is created, the class constructor function is automatically called.
- The constructor is always named `__init__()`
- It contains code for initializing a new instance of the class to a specific initial state (e.g. setting instance attribute values).

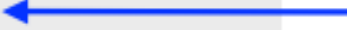
```
>>> class Person:
...     def __init__( self, s ):
...         self.name = s
...
...     def hello( self ):
...         print "Hello", self.name
```

Constructor function taking
initial value for instance
attribute name



```
>>> t = Person("John")
>>> t.hello()
Hello John
```

Calls `__init__()`
on Person



Example

Following is the example of a simple Python class –

```
class Employee:
    'Common base class for all employees'
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

    def displayCount(self):
        print "Total Employee %d" % Employee.empCount

    def displayEmployee(self):
        print "Name : ", self.name, " , Salary: ", self.salary
```

- The variable *empCount* is a class variable whose value is shared among all instances of a this class. This can be accessed as *Employee.empCount* from inside the class or outside the class.
- The first method `__init__()` is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
- You declare other class methods like normal functions with the exception that the first argument to each method is *self*. Python adds the *self* argument to the list for you; you do not need to include it when you call the methods.

Creating Instance Objects

To create instances of a class, you call the class using class name and pass in whatever arguments its `__init__` method accepts.

```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

Accessing Attributes

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows –

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```