* id() function -- returns a unique identifier for an object. It deals with the object not the reference.
* Value equality means that two variables have same value
* Identity equality means that two variables reference same object.
* During pass by object reference -- the value of the reference is copied not the value of the object.This basically mean that when the variable is passed as argument to the function it carries the value of the object it points towards in memory and this memory value is passed on not the value inside the object.

E.g.,

Def f(d)

Print id(d)

Return d

C =[1,2,3]

E = f(c)

C is e will be true

* The + operator will concatenate two or more strings - variables or literals
* List -- a mutable collection of any type. Used to represent mutable sequence of elements.
* Tuple -- an immutable collection
* Range -- immutable collection of integers
* Dictionary -- effectively a hash table
* If we assign a function to another variable than it means second variable will perform same function as first
  + def f(d):
  + ... print id(d)
  + ... return d

Now

f(2) will print two values (8420432 & 2)

v =f

v(2) will print two values as well

m=f(2) will print 8420432

The lack of parentheses signals that we don't want to assign the result of calling the function but instead assign the function itself. This gives us access to the same functionality through a new name.

* A function can return only one value if you required to return more than one value an aggregate such as a list or dictionary can be returned.
* Naming in python:
  + Module\_name, package\_name, ClassName, method\_name, ExceptionName, function\_name, GLOBAL\_CONSTANT\_NAME, global\_var\_name, instance\_var\_name, function\_parameter\_name, loca\_var\_name
* Dictionaries support the keys() and values() methods to give access to the contents as lists
  + D = {‘three’:3, ‘one’:1}
  + D.keys() --- [‘three’,’one’]
  + D.values() --- [3,1]
  + The items() return the key,value pairs as a list of tuples. E.g., [(‘three’,3),(‘one’,1)]
* Tuples are similar to lists in their structure and construction although they are immutable. Nesting, slicing and accessing elements by position are supported but there are no methods to alter the tuple’s content.
  + Tup1 = (1,3,5)
  + Tup1[2] ---- 5
* Python allows functions to accept named parameters as a sequence of name=value pairs. In the function definition we provide the name of a dictionary using the syntax \*\*dictname and python will package the named arguments into this dictionary on function call.
  + Def named (a,b,\*\*params):

Print params

named(1,2,three=3,four=4)

{‘three’:3,’four’:4}

* When we wish to pass a tuple or list to a function we define it using the form \*listOrTupleName. E.g.,

Def f (\*args):

Print “args:”, args

f(0,1,2,3,4,5)

Args: (0,1,2,3,4,5)

**Object Oriented Programming Python:**

Programs built as collections of objects -- these object interact with each other by sending messages, allow users by developing more realistic modelling of real world.

Objects are described by classes. Define which messages can be received using methods.

A class define the state and behaviour. The \_\_init\_\_ method can be used to initialise new objects and can be thought of as a constructor.

An important point to note is that python has no support for data encapsulation, which means the variables within a class can be accessed directly without using class.

self :self represents the instance of the class. By using the "*self*" keyword we can access the *attributes* and *methods* of the class in python.

\_\_init\_\_ : "\_\_init\_\_" is a reserved method in python classes. It is known as a constructor in object oriented concepts. This method called when an object is created from the class and it allow the class to initialize the attributes of a class.

*"self"* represents the same object or instance of the class. "self" represents the object inside the class.we have used "self" as a parameter in the method because whenever we call the method the object (instance of class) automatically passes as a first argument along with other argumets of the method.If no other arguments are provided only "self" is passed to the method. **E.g.,** we use object( instance of class) to call the method outside of the class definition("r.get\_area()"). "r" is the instance of the class, when we call method "r.get\_area()" the instance "r" is passed as as first argument in the place of self.

Python represents objects as areas of dynamically allocated memory accessed through reference variables.

Every individual object has a unique internal identifier representing the identity of the object. You can use the id() function to confirm the results of the assignment.

Class data is data that is shared among all objects created. It can be called global data or variable.

@staticmethod decorator is used to make a method static ie it does not have a self argument added as it is invoked on the class although we can invoke it on an instance of class.

One unusual aspect of objects in python is the ability to add attributes to an object after it has been created even add methods. This means that although every object will have the attributes and methods defined in the class additional data or methods can be added as well. Only exception is that the post created method should be called both relative to the object and with the object passed as its first argument.

**Example:**

|  |
| --- |
| #!/usr/bin/python  class Account(object):  num\_accounts = 0  def \_\_init\_\_(self, balance):  self.balance = balance  Account.num\_accounts += 1  def deposit(self,amount):  self.balance += amount  def withdraw(self,amount):  self.balance -= amount  def get\_balance(self):  return self.balance  @staticmethod  def how\_many\_accounts():  return Account.num\_accounts  a1 = Account(0)  a2 = Account(300)  a1.deposit(10)  a2.withdraw(40)  a2.deposit(400)  a1.acc\_number = 1234  a1.show\_number = lambda acc: acc.acc\_number  print "a1 Balance is: ", a1.get\_balance()  print "a2 Balance is: ", a2.get\_balance()  a3 = Account(0)  a2 = a3  print ""  print "a3 Balance is: ", a3.get\_balance()  print "a2 Balance is: ", a2.get\_balance()  print ""  print "a1 id is:", id(a1)  print "a2 id is:", id(a2)  print "a3 id is:", id(a3)  print ""  print "we have total number of " + str(Account.how\_many\_accounts()) + " accounts."  print ""  print "Account a1 has number", a1.show\_number(a1)  print "Account a2 has number", a2.show\_number(a2) |

**Import specific type from a module:** From bank.account import Account. E.g., Account(100)

**Import and rename a module:** import bank.account as acc. E.g., acc.Account(100)

**Import and rename a type from a module:** from bank.account import Account as Acc. E.g., Acc(100)

Python searches number of directories when resolving package names to directories during import statements. The directories are held in internal list known as sys.path.

**Every class and object has a set of attributes set by default.**

**For classes:**

\_\_name\_\_ : the name of the class

\_\_module\_\_: the module from which it was loaded

\_\_bases\_\_ : a tuple of its base classes

\_\_dict\_\_ : a dictornary containing all the attributes (including methods)

\_\_doc\_\_: the documentation string

**For Objects:**

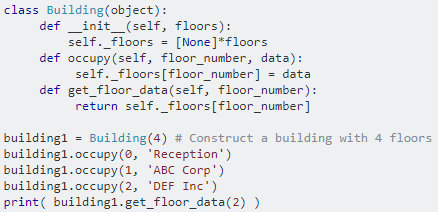
\_\_class\_\_ : the name of the class of the object

\_\_dict\_\_ : a dictionary containing all the object’s attributes

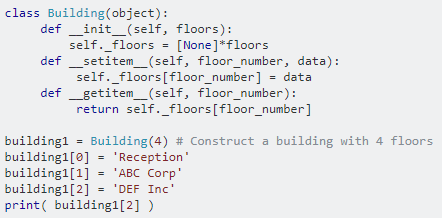
**\_\_get\_\_ & \_\_set\_\_ methods:**

Imagine a class which models a building. Within the data for the building it includes a number of attributes, including descriptions of the companies which occupy each floor :

Without using \_\_getitem\_\_ we would have a class like this :



We could however use \_\_getitem\_\_ (and it's counterpart \_\_setitem\_\_) to make the usage of the Building class 'nicer'.



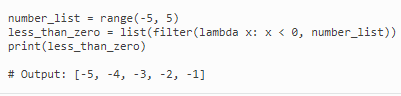
Whether you use \_\_setitem\_\_ like this really depends on how you plan to abstract your data - in this case we have decided to treat a building as a container of floors (and you could also implement an iterator for the Building, and maybe even the ability to slice - i.e. get more than one floor's data at a time - it depends what you need.

**MAP** applies a function to all the items in an input\_list. map(function\_to\_apply, list\_of\_inputs)

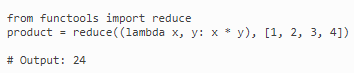
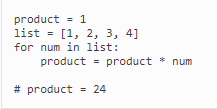
***fun :*** *It is a function to which map passes each element of given iterable.*

***iter :*** *It is a iterable which is to be mapped*

**FILTER** creates a list of elements for which a function returns true.



**REDUCE** is a really useful function for performing some computation on a list and returning the result. It applies a rolling computation to sequential pairs of values in a list.



Decorators are functions which modify the functionality of another function.They wrap a function and modify its behaviour in one way or the another. E.g.:

**ITERATOR:**

There are 3 parts to iterators:

1. Iterable: is any object in Python which has an \_\_iter\_\_ or a \_\_getitem\_\_ method defined which returns an **iterator** or can take indexes
2. Iterator: An iterator is any object in Python which has a next (Python2) or \_\_next\_\_ method defined.
3. Iteration: it is the process of taking an item from something e.g a list.

**Generators** are iterators, but you can only iterate over them once. It’s because they do not store all the values in memory, they generate the values on the fly. You use them by iterating over them, either with a ‘for’ loop or by passing them to any function or construct that iterates.

we are going to define a generator using a function with the **yield** statement/keyword. The **yield** instruction should be put into a place where the generator returns an intermediate result to the caller and sleeps until the next invocation occurs.

Next method steps through collections.

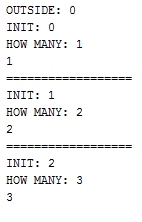
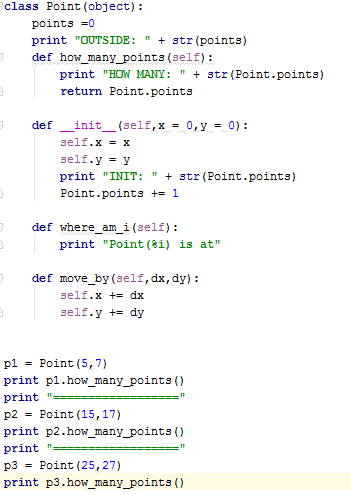
Xrange and readlines generate iterators which supply a single number/line on demand -not up front.

In python all classes have a single root or superclass **OBJECT** but can inherit from multiple classes.

Python use object references to point to objects, these references are stored on the stack and managed by the virtual machine. When the reference count of an object becomes 0 the VM automatically removes that object with a process of garbage collection.

Class attributes are called static data in other languages these are the variables defined outside all the functions and at class level. There value belong to all objects and shown by all.once initialized and then value incremented for class variable will hold the new value for new objects as well created later on.

Example:



All python classes have 5 intrinsic attributes that you can use freely in your code.:

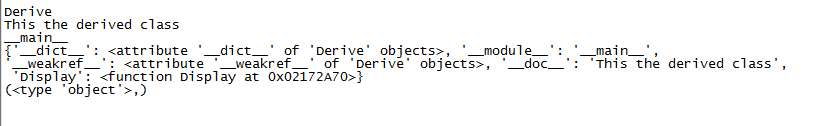
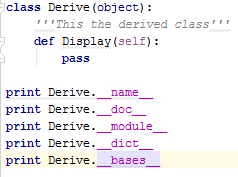
\_\_name\_\_ : name of the class (read only)

\_\_doc\_\_ : class documentation string

\_\_module\_\_ : module in which the class is defined

\_\_dict\_\_ : map of all class attributes

\_\_bases\_\_ : tuple of all base classes



Operator overloading allows you to define functions that define what happens to your objects when combined with the standard operators such as +-\*/. Defining methods for operators is known as operator overloading

Objects are an encapsulation of variables and functions into a single entity. Objects get their variables and functions from classes. Classes are essentially a template to create your objects

**===================PYTHON-3 UDEMY COURSE=======================**

**===================SECTION-1=======================**

<http://py3readiness.org/>

<https://pythonclock.org/>

<http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html>

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**

**===================SECTION-1=======================**