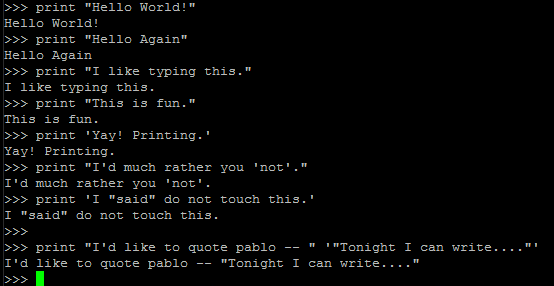
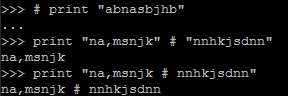
**EX 1: First Hello**

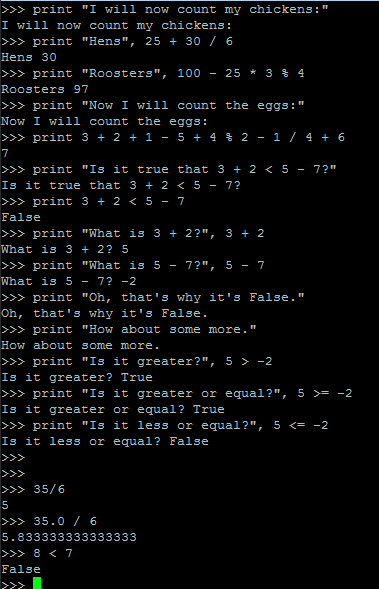


**Note : Wrap with “” to use ‘ and vv**

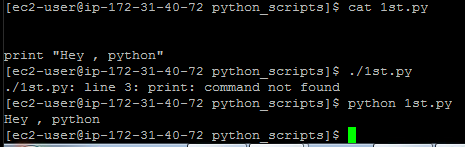
**Ex 1.1: Commenting out**

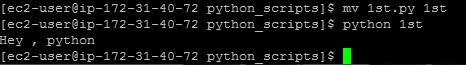


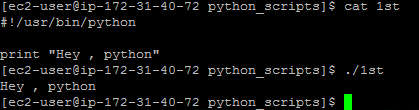
**Ex 2 : Numbers and Math**



**Ex 3 : Your first script**

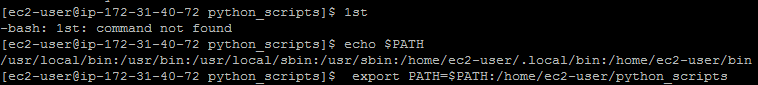






**Note : Update PATH variable with script location**

**Then you can run script directly like**





**Ex 4: variable**

cars **= 100**

space\_in\_a\_car **= 4.0**

drivers **= 30**

passengers **= 90**

cars\_not\_driven **=** cars **-** drivers

cars\_driven **=** drivers

carpool\_capacity **=** cars\_driven **\*** space\_in\_a\_car

average\_passengers\_per\_car **=** passengers **/** cars\_driven

**print** "There are"**,** cars**,** "cars available."

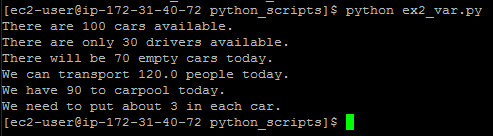
**print** "There are only"**,** drivers**,** "drivers available."

**print** "There will be"**,** cars\_not\_driven**,** "empty cars today."

**print** "We can transport"**,** carpool\_capacity**,** "people today."

**print** "We have"**,** passengers**,** "to carpool today."

**print** "We need to put about"**,** average\_passengers\_per\_car**,** "in each car."



**Note : Printing variables : Separate them with , from the quote**

**Ex 4.1 : More variables and printing**

my\_age **= 35** *# not a lie*

my\_height **= 74** *# inches*

my\_weight **= 180** *# lbs*

my\_eyes **=** 'Blue'

my\_teeth **=** 'White'

my\_hair **=** 'Brown'

**print** "Let's talk about %s." **%** my\_name

**print** "He's %d inches tall." **%** my\_height

**print** "He's %d pounds heavy." **%** my\_weight

**print** "Actually that's not too heavy."

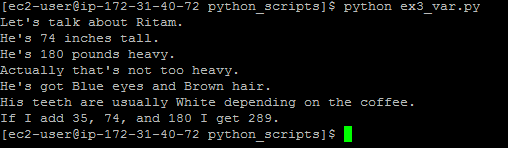
**print** "He's got %s eyes and %s hair." **% (**my\_eyes**,** my\_hair**)**

**print** "His teeth are usually %s depending on the coffee." **%** my\_teeth

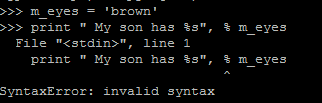
*# this line is tricky, try to get it exactly right*

**print** "If I add %d, %d, and %d I get %d." **% (**

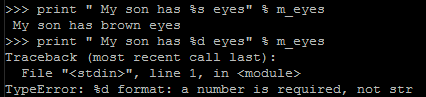
my\_age**,** my\_height**,** my\_weight**,** my\_age **+** my\_height **+** my\_weight**)**



**Extrass:**



Don’t use , before %



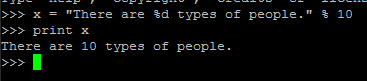
%s 🡪 string %d 🡪 Number

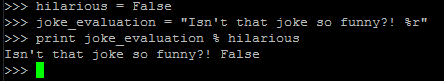


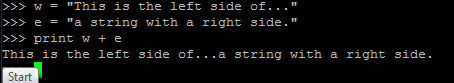
Use the **%r** for debugging, since it displays the **"raw" data of the variable**, but the others are used for displaying to

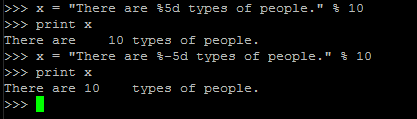
users.

**Ex 4.3 More printing examples**







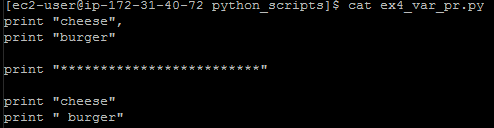


**%5d : puts 5 spaces before**

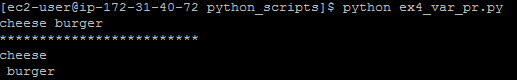
**%-5d: puts 5 spaces after**

**EX 4.4: use of , between 2 print func**

**Scr:**



**o/p:**



**Ex 4.5: Formatter**

formatter **=** "%r %r %r %r"

**print** formatter **% (1, 2, 3, 4)**

**print** formatter **% (**"one"**,** "two"**,** "three"**,** "four"**)**

**print** formatter **% (**True**,** False**,** False**,** True**)**

**print** formatter **% (**formatter**,** formatter**,** formatter**,** formatter**)**

**print** formatter **% (**

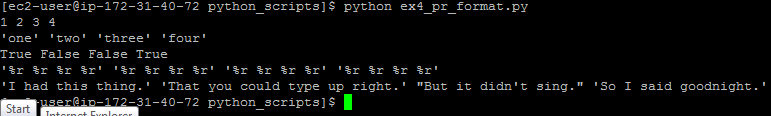
"I had this thing."**,**

"That you could type up right."**,**

"But it didn't sing."**,**

"So I said goodnight."

**)**



**Ex 4.6: 3 double quotes and newline etc**

**Scr:**

*# Here's some new strange stuff, remember type it exactly.*

**days = "Mon Tue Wed Thu Fri Sat \tSun"**

**months = "\nJan\nFeb\nMar\nApr\nMay\nJun\nJul\nAug"**

**print "Here are the days: ", days**

**print "Here are the months: ", months**

**print """**

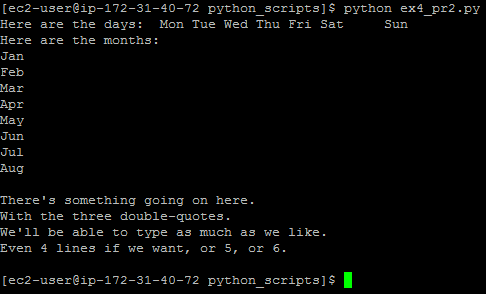
**There's something going on here.**

**With the three double-quotes.**

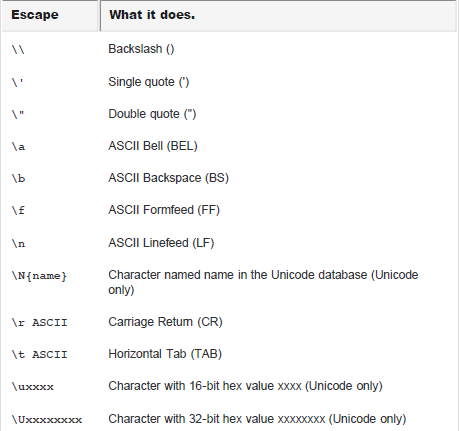
**We'll be able to type as much as we like.**

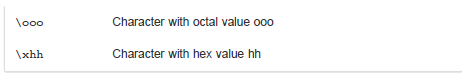
**Even**4 lines if we want, or 5, or 6.

"""

**o/p:** 

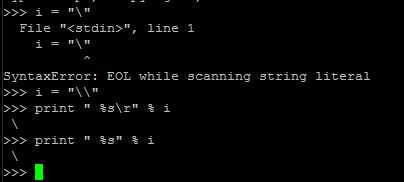
**Ex 4.7 : Escape Sequences**





**Escaping single or double quote**





**Ex 5: raw\_input and input func**

**print "How old are you?",**

**age = raw\_input()**

**print "How tall are you?",**

**height = raw\_input()**

**print "How much do you weigh?",**

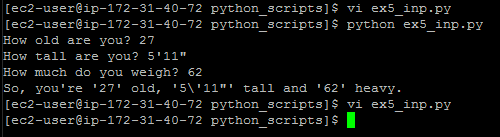
**weight = raw\_input()**

**print "So, you're %r old, %r tall and %r heavy." % (**

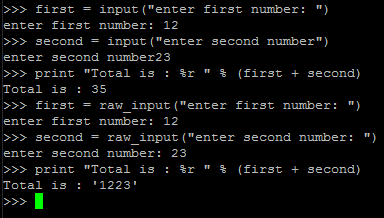
**age, height, weight)**

Getting input from

User.



**5.1 : Difference between input() raw\_input()**



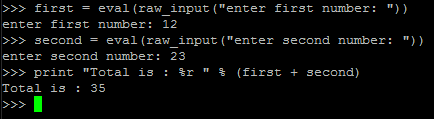
**Note: \***

**input evaluates user input and converts it to int , float or string**

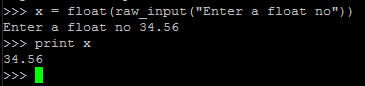
**Raw\_input returns string only**

**\*\* you can print message for user inside braces like –**

**5.2 : How to do math ops using raw\_input**



**Note: In place of eval we can also use int, float etc to convert the input string**



**Ex 6 : Parameters,**

**from sys import argv**

**script, first, second, third = argv**

**print "The script is called:", script**

**print "Your first variable is:", first**

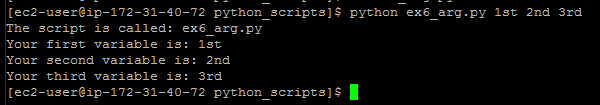
**print "Your second variable is:", second**

**print "Your third variable is:", third**

**Unpacking, Variables**

**Passing variables in script**

**Script 🡪**



**"import". This is how you add features to your script from the Python feature set.**

**Rather than give you all the features at once, Python asks you to say what you plan to use. This keeps your programs small.**

**argv is the "argument variable",**

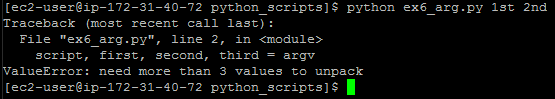
**This variable holds the arguments you pass to your Python script when you run it.**

**unpacks" argv so that, rather than holding all the arguments, it gets assigned to four variables you can work**

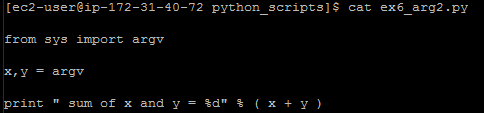
**with: script, first, second, and third.**

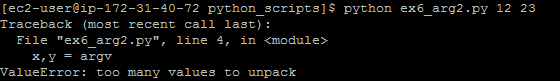
**Issues:**

* **Not enough Arguments**



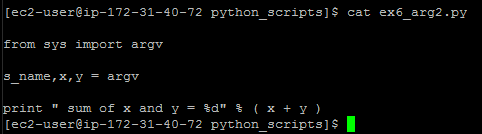
**\*\* Too many values**

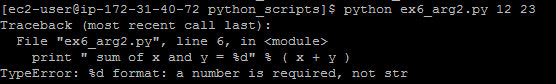
 **scr**

 **o/p**

**Note : script name is by default the 1st args**

**\*\*\***

 **scr**

 **o/p**

**Note: Argv takes the values as strings .**

**Convert them to int/float to use math operations.**

**Ex 7 : Reading Files**

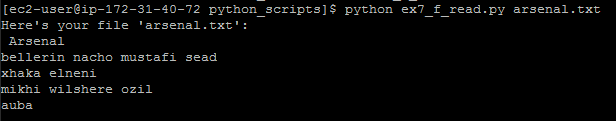
**from sys import argv**

**script, filename = argv**

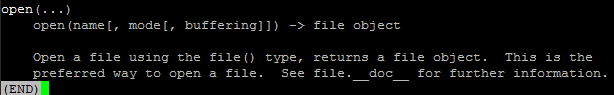
**txt = open(filename)**

**print "Here's your file %r:" % filename**

**print txt.read()**



**Note:**

****

**Read , Readline :**

**read(...)**

**| read([size]) -> read at most size bytes, returned as a string. If the size argument is negative or omitted, read until EOF is reached.**

**| Notice that when in non-blocking mode, less data than what was requested. may be returned, even if no size parameter was given.**

**| readline(...)**

**| readline([size]) -> next line from the file, as a string.**

**| Retain newline. A non-negative size argument limits the maximum**

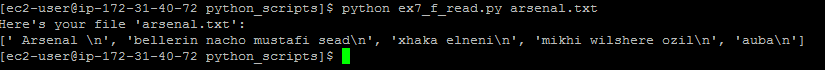
**| number of bytes to return (an incomplete line may be returned then).**

**| Return an empty string at EOF.**

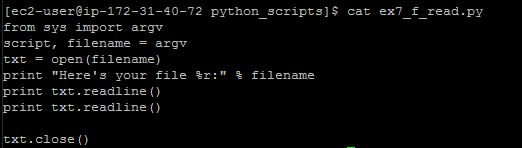
**| readlines(...)**

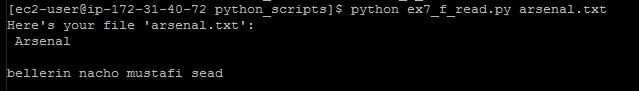
**| readlines([size]) -> list of strings, each a line from the file.**

* **Using readlines() in place of print txt.read()**



**\*\* Using readline()**

**Scr 🡪** 

**o/p**🡪 

**Have your script also do a close() on the txt and txt\_again variables. It's important to close files when you are done with them.**

**Ex 8 : Writing into a file**

**from sys import argv**

**from os.path import exists**

**script, from\_file, to\_file = argv**

**print "Copying from %s to %s" % (from\_file, to\_file)**

***# we could do these two on one line too, how?***

***#indata = open(from\_file).read()***

**in\_file = open(from\_file)**

**indata = in\_file.read()**

**print "The input file is %d bytes long" % len(indata)**

**print "Does the output file exist? %r" % exists(to\_file)**

**print "Ready, hit RETURN to continue, CTRL-C to abort."**

**raw\_input()**

**out\_file = open(to\_file, 'w')**

**out\_file.write(indata)**

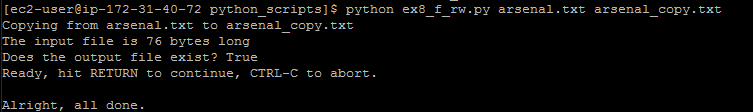
**print "Alright, all done."**

**out\_file.close()**

**in\_file.close()**

**In\_file**🡪 opens a file object for from\_file

We can also use like  
indata = open(from\_file).read()



**Note : indata = in\_file.read(20)**

**Allows only first 20 bytes of input file to be read.**

**Ex8.1 : Deleting a file content and writing again**

**from sys import argv**

**script, filename = argv**

**print "We're going to erase %r." % filename**

**print "If you don't want that, hit CTRL-C (^C)."**

**print "If you do want that, hit RETURN."**

**raw\_input("?")**

**print "Opening the file..."**

**target = open(filename, 'w')**

**print "Truncating the file. Goodbye!"**

**target.truncate()**

**print "Now I'm going to ask you for three lines."**

**line1 = raw\_input("line 1: ")**

**line2 = raw\_input("line 2: ")**

**line3 = raw\_input("line 3: ")**

**print "I'm going to write these to the file."**

**target.write(line1)**

**target.write("\n")**

**target.write(line2)**

**target.write("\n")**

**target.write(line3)**

**target.write("\n")**

**print "And finally, we close it."**

**target.close()**

**Truncates the file, ‘target’ holds**

**Remember:**

**1st , open a file object**

**To read🡪**

**F\_obj.read(size in byte)**

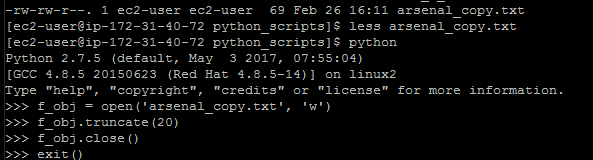
**Gives str o/p of file data**

**To Write 🡪**

**F\_obj.write(strings to be written)**

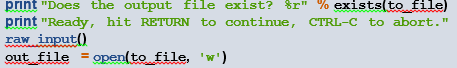
**To delete content 🡪**

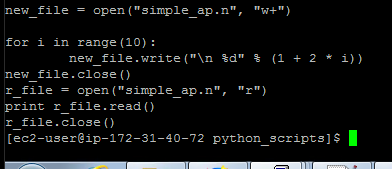
**F\_obj.truncate()**

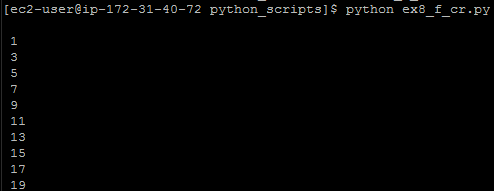


**Truncates the file to 20 bytes but makes it a binary file.**

**Ex 8.2: Creating a file does not exists**

**Remember :** 

**scr**

**o/p**

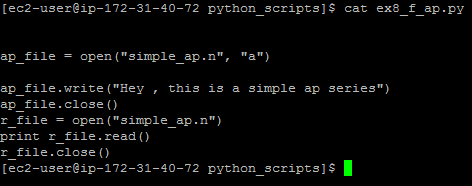
**Note : sytax : open(“f\_name”, “w/w+/r/a”)**

**+ sign indicates , if the file does not exists , it will create it.**

**Other file modes : 'x' Creates a new file. If file already exists, the operation fails.**

**‘b’ 🡪 binary**

**‘t’ 🡪 default mode , opens in text mode.**

**Ex 8.3: Append**

**EX 9: Function**

**9.1: 1st function definition and call**

**don’t forget :**

**def hello(inp):**

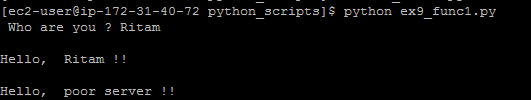
**print "Hello, ", inp "!!"**

**text = raw\_input(" Who are you?")**

**hello(text)**

**hello("poor server")**

* **‘:’ after func definition**
* **Indentation  
  we don’t have a brace to close. It’s the indentation that’s separates it.**



**Note : Look how a variable (without’’) and raw text is passed**

**9.2: Passing multiple arguments**

**def sum(\*args):**

**a1, a2, a3 = args**

**return (a1 + a2 + a3)**

**def mult(a1,a2,a3):**

**return (a1 \* a2 \*a3)**

**sum(12,34, 8)**

**mult(12,34,8)**



**Ex 9.3: Passing function as parameter**

**def minmax(test, \*args):**

**print test**

**res = args[0]**

**for arg in args[1:]:**

**if test(arg, res):**

**res = arg**

**return res**

**def lessthan(x, y): return x < y**

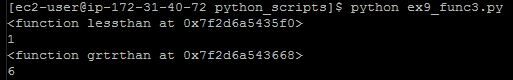
**def grtrthan(x, y): return x > y**

**print minmax(lessthan, 4, 2, 1, 5, 6, 3)**

**print minmax(grtrthan, 4, 2, 1, 5, 6, 3)**

**Lessthan and grtrthan func**

**Returns Boolean .**



**EX 9.4: Recursive**

**def factorial(n):**

**if n == 1:**

**return 1**

**else:**

**return n\*factorial(n-1)**

**print factorial(10)**

**Ex 9.5 Lambda function**

T**he lambda operator or lambda function is a way to create small anonymous functions, i.e. functions without a name. These functions are throw-away functions, i.e. they are just needed where they have been created. Lambda functions are mainly used in combination 🡪 with the functions *filter(), map() and reduce().***

**def f (x): return x\*\*2**

**print f(8)**

**g = lambda x: x\*\*2**

**print g(8)**

**Ex 9.6 Default argument:**

**9.6.1: Making argument global and not making it**

**#Global**

**def f(x, y=[]):**

**y.append(x)**

**return y**

**print f(23)**

**print f(19)**

**#Noo Global**

**def f2(x, y=None):**

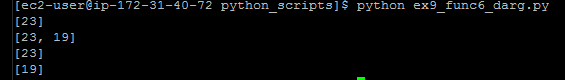
**if y is None: y = []**

**y.append(x)**

**return y**

**print f2(23)**

**print f2(19)**



**Ex: 9.6.2: More example**

**def ask\_ok(prompt, retries=4, complaint='Yes or no, please!'):**

**while True:**

**ok = raw\_input(prompt)**

**if ok in ('y', 'ye', 'yes'): return True**

**if ok in ('n', 'no', 'nop', 'nope'): return False**

**retries = retries - 1**

**if retries < 0: raise IOError, 'refusenik user'**

**print complaint**

**c = ask\_ok('Do you really want to quit?')**

**if c is True:**

**print " quiting now..."**

**elif c is False:**

**print " continue..."**

**We will take a break here and come back to function after list and dictionary**

**Ex:10 Importing your script and some cool string operations**

**def break\_words(stuff):**

**"""This function will break up words for us."""**

**words = stuff.split(' ')**

**return words**

**def sort\_words(words):**

**"""Sorts the words."""**

**return sorted(words)**

**def print\_first\_word(words):**

**"""Prints the first word after popping it off."""**

**word = words.pop(0)**

**print word**

**def print\_last\_word(words):**

**"""Prints the last word after popping it off."""**

**word = words.pop(-1)**

**print word**

**def sort\_sentence(sentence):**

**"""Takes in a full sentence and returns the sorted words."""**

**words = break\_words(sentence)**

**return sort\_words(words)**

**def print\_first\_and\_last(sentence):**

**"""Prints the first and last words of the sentence."""**

**words = break\_words(sentence)**

**print\_first\_word(words)**

**print\_last\_word(words)**

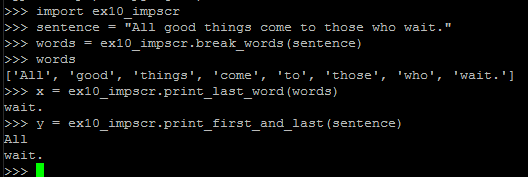
**def print\_first\_and\_last\_sorted(sentence):**

**"""Sorts the words then prints the first and last one."""**

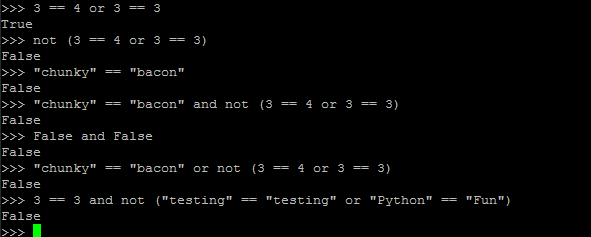
**words = sort\_sentence(sentence)**

**print\_first\_word(words)**

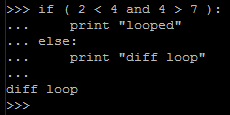
**print\_last\_word(words)**



**Ex 11 : Some Boolean Practice**



**Ex 12: If elif else : loop**



**Syntax:**

**If condition :**

**Statement and commands  
elif condition:**

**Statement and commands**

**New loop :   
 Statement and commands of new loop**

**Else:**

**Statement and commands**

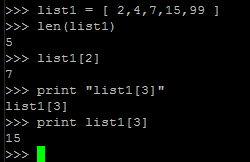
**Note : careful with the indentation , we don’t have any braces opening for a loop**

**‘:’🡪 indicates loop initialization**

**‘\t’ 🡪 separates it from the rest of the code**

**Ex 12: List**

**Ex12.1: Creating list**

 **a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.**

* **Accessing a single element**
* **Don’t make that mistake**



**Ex 12.2 inserting elements into a list :**

**Intro of for loop**

**fibo = []**

**n0 = 1**

**n1 = 1**

**fibo.append(n0)**

**fibo.append(n1)**

**for i in range(2, 9):**

**n\_temp = n1**

**n1 = n1 + n0**

**n0 = n\_temp**

**fibo.append(n1)**

**print fibo**

* **list.append(‘string/number/  
  variable)**



**12.2.1 For Loop**

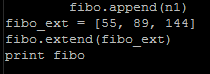
**FOR Loop  
styles:**

**#*count in range from a to b*  
For count in range(a, b):**

***#count in range from 0 to n*  
For count in range(n):**

***#count in each and every element of a list*  
for count in list**

**Ex 12.3: Extending the list**

**scr**

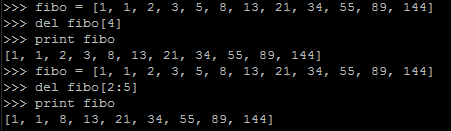
 **o/p**

**Ex 12.4 Removing from list**

**12.4.1: using del**

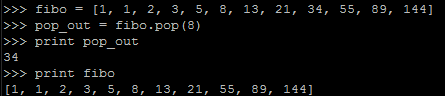
**USING DEL: syntax : del list[index]**

**Del list[Ind+low\_lim:ind\_up\_lim]**



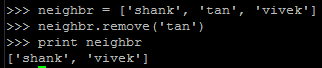
**12.4.2: using pop**

**Syntax: list.pop(index)**

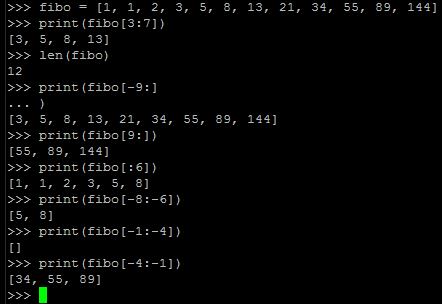


**12.4.3: using remove**

**Syntax: list.remove(element name)**

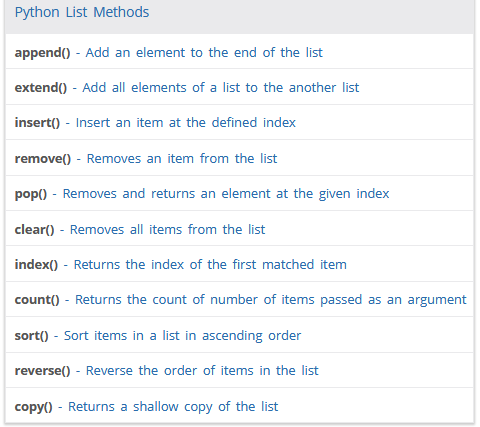


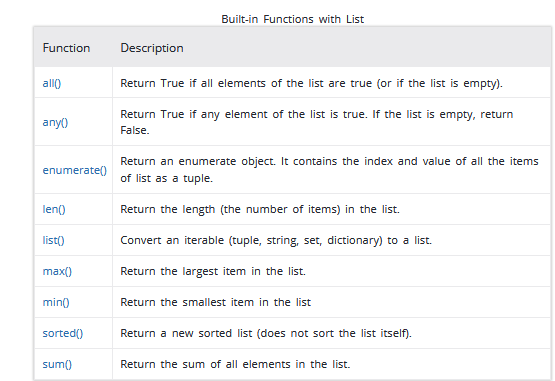
**Ex12.5: Slicing the list**

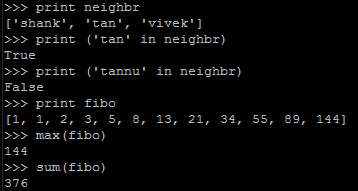
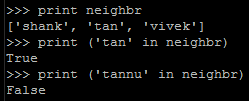
**python allows**

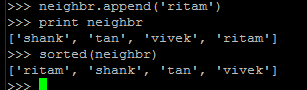
**Ex12.6: Insertion**

**List Methods**









**Ex 13: Introduction of While loop**

**num = int(raw\_input("Enter the no to check if Armstrong >"))**

**p\_num = num**

**s = 0**

**while num > 0:**

**temp = num % 10**

**s += temp \*\* 3**

**num //= 10**

**print "Sum is " , s**

**if num == s:**

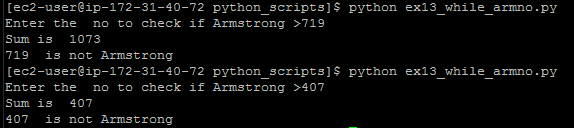
**print p\_num, " is a Armstrong Number"**

**else:**

**print p\_num, " is not Armstrong"**

**While condition :**

**Code (if true)**



**Ex 14 Break and continue**

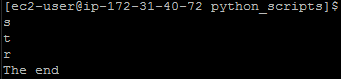
**for val in "string":**

**if val == "i":**

**break**

**print(val)**

**print("The end")**



**Ex14.1: Continue:**

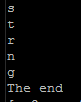
**for val in "string":**

**if val == "i":**

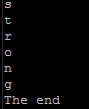
**continue**

**print(val)**

**print("The end")**

 **o/p**

**If we change the word as “strong”**

**Look print(val) is the important part of the script.**

**If condition == True 🡪 loop will continue for next iteration and code after continue will not be executed.**

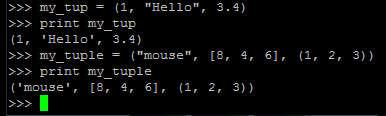
**Pass statement:**

**In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored.**

**However, nothing happens when pass is executed. It results into no operation (NOP).**

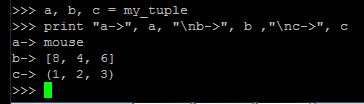
**Ex15: Introduction of Tuples:**

**a tuple is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed.**

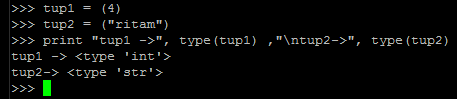
**Remember: in list 🡪 use []**

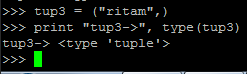
**Tuple 🡪 ()**

**Ex15.0.1:Unpacking tuples:**



**Ex 15.0.2: Creating tuples of single element:**

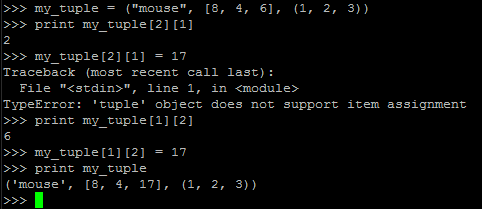
**() is not enough**

**the’,’ is important**

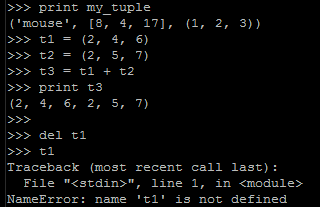
**indexing and slicing similar to list**

**Ex 15.1: changing tuples:**

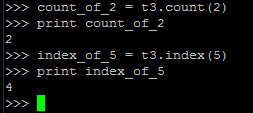
**tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed.**



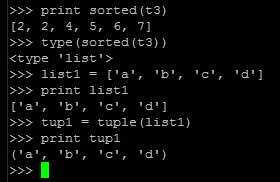
**Ex 15.2: concat and del**



**Ex15.3: count and index:**

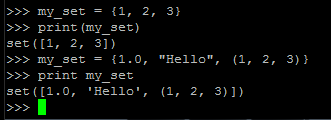


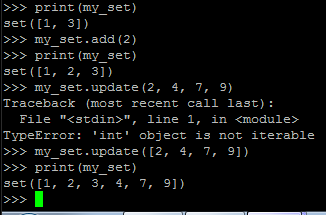
**Ex 15.4: some more func:**



**Ex 16 : Introduction to SET:**

A set is an unordered collection of items. Every element is unique (**no duplicates**) and must be **immutable** (which cannot be changed).

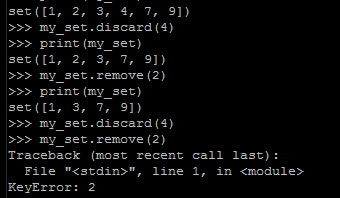
**Ex 16.1 set creation**

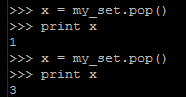
**Ex 16.2 change a set:**

**set.add(single\_element)**

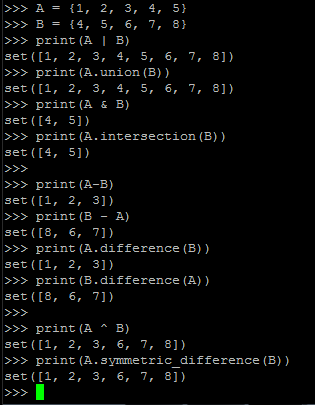
**set.update(list or tuple or set)**

**Ex 16.3 deleting an element:**





**Ex 16.4: Set operations :**

**|**

**| 🡪 Union**

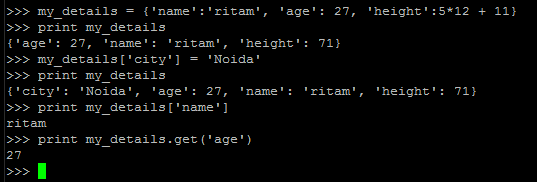
**& 🡪 Intersection**

**‘-‘ --> difference**

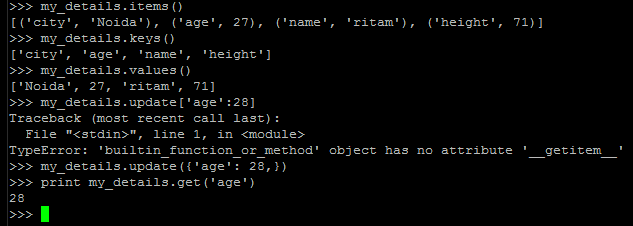
**^ 🡪 Symmetric difference**

**Ex17 Introduction to dictionary:**

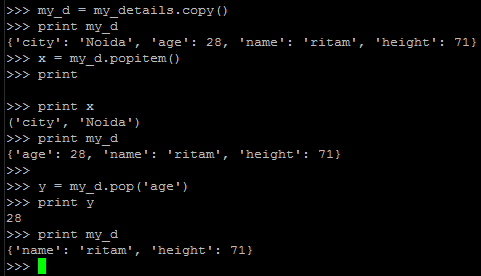
**Ex17.1: adding creating and retrieving**



**Ex 17.2 : dictionary methods:**

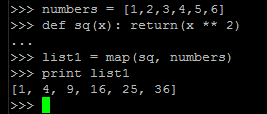


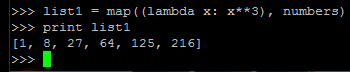
**Ex 17.3 : Removing**



**Ex 18 : Expression oriented functions**

**Ex18.1: map(aFunction, aSequence)**



**Ex18.1.1: using lambda functions**

**Ex 18.1.2 Another example**

**def square(x):**

**return (x\*\*2)**

**def cube(x):**

**return (x\*\*3)**

**def fact(x):**

**if x == 0 :**

**return x**

**if x == 1 :**

**return x**

**result = x**

**while x > 1:**

**result \*= (x -1)**

**x -= 1**

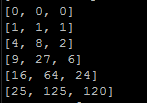
**return result**

**funcs = [square, cube, fact]**

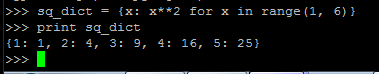
**for r in range(6):**

**value = map(lambda x: x(r), funcs)**

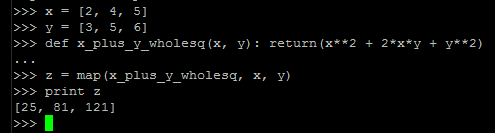
**print value**

We are using **lamda as a aFunction**, **we can have a list of functions as aSequence:** 

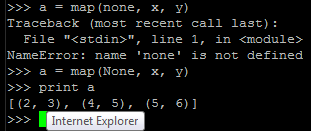
**Ex 18.2 creating dict , another method (not a map example):**



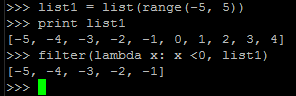
**Ex 18.1.3:map() expects an N-argument function for N sequences.**



**Ex 18.1.4:** If **function is None**, the **identity function is assumed**; if there are multiple arguments, **map() returns** a **list** **consisting of tuples** containing the corresponding items from all iterables (a kind of transpose operation). The iterable arguments may be a sequence or any iterable object; the result is always a list:

**its ‘None’**

**Ex 18.3 : Filter**

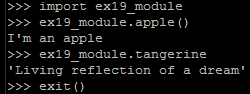


**Ex 19: Modules**

**def apple():**

**print("I'm an apple")**

**tangerine = "Living reflection of a dream"**



**Ex 20 : class**

**Class :**

Class is a set or category of things having some property or attribute in common and differentiated from others by kind, type, or quality.

In technical terms we can say that class is a blue print for individual objects with exact behaviour.

**Object :**

object is one of **instances of the class.** which can perform the functionalities which are defined in the 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 **reseved 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.**

**Ex20.1**

**class first\_class:**

**"This is my 1st class in python"**

**var = 100**

**def hello(self):**

**print "Hello Python!!"**

**print first\_class.var**

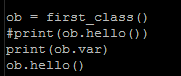
**print(first\_class.hello)**

**print first\_class.\_\_doc\_\_**



**Ex20.2**

**Creating class object**

 **🡨 snippet**

 **🡨 o/p**

**Ex20.3: \_\_init\_\_**

**class complex\_number:**

**"this will create a complex number"**

**def \_\_init\_\_(self, r = 0, i = 0):**

**self.real = r**

**self.img = i**

**def getdata(self):**

**print("{0} + i{1}".format(self.real, self.img))**

**c1 = complex\_number(2,5)**

**c1.getdata()**

**c2 = complex\_number(0, -7)**

**c2.attr = '2nd'**

**c2.getdata()**

**print "c2 --> ", c2.attr**

**print "c1 --> ", c1.attr**

**To pass value and collect**

**Use of format func**

**C1.attr will throw an error**

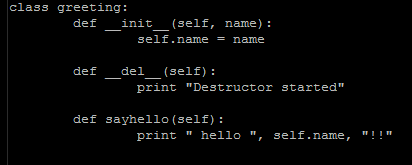
**But attributes of the objects can be created on the fly.**

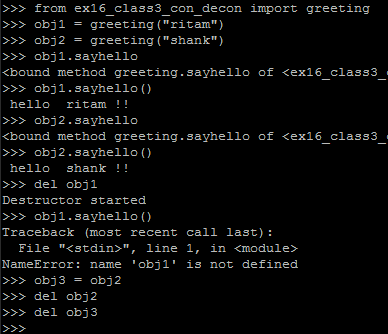
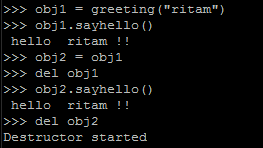
**To delete attribute : del c1.img**

**To delete object : del c1**

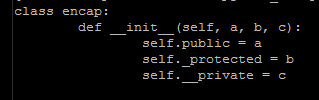
**EX 20.3: Constructor and deconstructor**

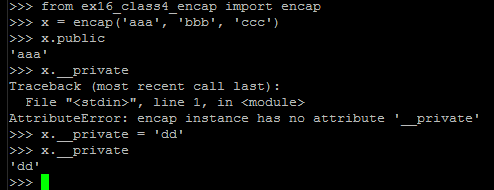
**\_\_init\_\_ & \_\_del\_\_**

 **class**



**Ex 20.4 : Encapsulation**





**\_\_ to start declaring a private variable.**

**These are accessible within the object only.**

**Ex20.5: polymorphism**

**Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).**

**Suppose, we need to color a shape, there are multiple shape option (rectangle, square, circle). However we could use same method to color any shape. This concept is called Polymorphism.**

**class rect:**

**def \_\_init\_\_(self, x, y):**

**self.len = x**

**self.wid = y**

**def area(self):**

**area = 2 \* (self.len + self.wid)**

**print " Area of rectangle : ", area**

**class circle:**

**def \_\_init\_\_(self, r):**

**self.rad = r**

**def area(self):**

**pi = 3.14**

**area = pi \* (self.rad \*\* 2)**

**print " Area of circle : ", area**

**circle1 = circle(5)**

**rect1 = rect(3, 5)**

**def area\_call(shape):**

**shape.area()**

**area\_call(circle1)**

**area\_call(rect1)**

**2 classes**

**Common method area  
different function**

**To enforce polymorphism🡪**

**Create a function area\_Call**

**Which takes an object**

**And calls the method area for each object.**

**Depending on the class object we can call methods of respective classes**



**Ex20 .6 : Inheritance:**

**class polygon:**

**def \_\_init\_\_(self, no\_of\_sides):**

**self.no\_of\_sides = no\_of\_sides**

**print " polygon is ready.."**

**self.sides = [0 for i in range(self.no\_of\_sides)]**

**def input\_sides(self):**

**self.sides = [float(raw\_input("Enter Side %d : " % (i + 1))) for i in range(self.no\_of\_sides)]**

**def disp\_sides(self):**

**for i in range(self.no\_of\_sides):**

**print "side ", i + 1 , " : ", self.sides[i]**

**def perim(self):**

**return sum(self.sides)**

**class triangle(polygon):**

**def \_\_init\_\_(self):**

**polygon.\_\_init\_\_(self, 3)**

**print " I m a trianle"**

**def area(self):**

**s = float(self.perim())/2**

**a, b, c = self.sides**

**area = (s \* (s - a) \* (s - b) \* (s - c)) \*\* .5**

**print " Area of triangle is : ", area**

**t = triangle()**

**t.input\_sides()**

**t.disp\_sides()**

**t.perim()**

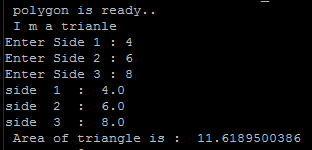
**t.area()**

**~**

**Class triangle is a polygon**

**T 🡪 object of triangle**

**Can access methods and attributes of polygon**



**Ex 20.6.1 super function**

**class Mammal(object):**

**def \_\_init\_\_(self, mammalName):**

**print(mammalName, 'is a warm-blooded animal.')**

**class Dog(Mammal):**

**def \_\_init\_\_(self):**

**print('Dog has four legs.')**

**super(Dog, self).\_\_init\_\_('Dog')**

**d1 = Dog()**

**Allows us to avoid using base class explicitly**

**Without using super func:**

**class Dog(Mammal):**

**def \_\_init\_\_(self):**

**print('Dog has four legs.')**

**Mammal.\_\_init\_\_(self, ‘Dog’)**

**With super()**

**We can easily change the base**

**Class for Dog.**