content Delivered in class_5_03-August-2016:

- Chapter 3: Language Components
  - Loops
    - for and while
  - break, continue, pass and sys.exit
- Chapter 4: Collections
  - Lists

Interview Questions Discussed

Interview Question 1: In which cases, list.extend is more suitable than list.append?

Interview Question 2: What is the sorting algorithm used in python?

Interview Question 3 : What is the difference between sort() and sorted()?

Assignments Given

Assignment 1: Run the below class5_multiplicationTables2.py, with the various commented options, in the script. Observe the differences.

Assignment 2: Based on class5_multiplicationTables2.py, write a program to display the multiplication tables from 1 through 10, horizontally

Assignment 3: Write a program to display a full diamond shape, separetely using for loop, and using while loop?

Assignment 4: Generate a Caesar Cipher for a given string using list comprehensions

Assignment 5: Pythogorean triples, between 1 and 55

Assignment 6: Implement Queue mechanism using lists (FIFO - Queue)

Assignment 5: If the temperature on 22 July, 2015 was recorded as 32°C, 44°C, 45.5°C and 22°C in Paris, Hyderabad, Visakhapatnam and Ontario respectively, what are their temperatures in °F.

Hint: T(°F) = T(°C) × 9/5 + 32

Assignment 6: If the temperatures on same day this year are forecasted as same numerics in °F, what will be their corresponding temperature in °C.

---

for Loop

General sematic for **for loop** in any language

for (initialization, condition, increment/decrement)

```
    logic
```

syntax in Python:

```
for indexingVaraible1, indexingVariable2,.. in (iterator1, iterator2,..):
    logic
```

In [1]:
```
for i in [1,2,3,5]:   # here, initialization is 1,
    print i
```

```
1
2
3
5
```

In [2]:
```
for i in range(2, 10, 2):     # initialization is 2, increment = 2, finalValue
 = 10
    print i,                  # , operator works as newline suppresser, here
```

```
2 4 6 8
```

In [3]:
```
for i in range(6):      # initialization = 0, increment = 1; finalValue = 6
    print i,
```

```
0 1 2 3 4 5
```

In [1]:
```
for i in 'Python':
    print i,
```

```
P y t h o n
```

In [3]:
```
words = ['Python', 'Programming', 'Django', 'NEtworking']
for i in words:
    print i,
```

```
Python Programming Django NEtworking
```

In Python, Pass, Continue and break are used to loops.

Though continue and break are similar to that of other traditional programming languages, pass is a unique feature available in python.

break - Terminates the loop, after its invocation.

continue- Skips the current loop, and continues perform the next consecutive loops.

pass - Does nothing. No logic to work. No break nor skipping a loop. pass is placed when there is no logic to be written for that condition, within the loop. It is advantageous for the developers for writing the logic in future.

In [4]:
```
for i in words:
    if i == 'Python':
        continue
    print i
```

```
Programming
Django
NEtworking
```

In [4]:
```
#!/usr/bin/python

#class5_passContinueBreakSysExit.py


print "\nExample to illustrate CONTINUE "
for j in range(10):
  if j==5:
    continue
  print j,

print "\nExample to illustrate BREAK "
for p in range(10):
  if p==5:
     break
  print p,

print "\nExample to illustrate PASS "
for i in range(10):
  if i==5:
     pass       # its like TODO
  print i,
```

```
Example to illustrate CONTINUE
0 1 2 3 4 6 7 8 9
Example to illustrate BREAK
0 1 2 3 4
Example to illustrate PASS
0 1 2 3 4 5 6 7 8 9
```

pass - It acts like a TODO during development. Generally, pass is used when logic will be written in future.

while loop

Syntax in python

```
initialization
while condition
    logic
    increment/decrement
```

Prefer while loop, only when the exit condition known; else it might lead to infinite loop.

In [5]:
```
a = 0                    # initialization
while a<20:              # condition
    print a,             # logic statement
    a+=1                 # increment
```

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

In [7]:
```
b = 34
while b>0:
    print b,
    b-=1
```

```
34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9
 8 7 6 5 4 3 2 1
```

In [8]:
```
# Fibonacci Series: 0,1,1,2,3,5,8,13,...
a,b = 0,1                # Tuple unpacking
while b < 100:           # to print fibonacci numbers less than 100
    print b,
    a,b = b, a+b
```

```
1 1 2 3 5 8 13 21 34 55 89
```

In [10]:
```
a,b = 0,1                # Tuple unpacking
count = 0       # initializing new variable
while b < 100:           # to print fibonacci numbers less than 100
    print b,
    a,b = b, a+b
    count+=1   # newVariables can't be used in loops, as new variable gets cre
ated everytime
```

```
 1 1 2 3 5 8 13 21 34 55 89
```

In [11]:
```
count
```

Out[11]:
```
11
```

In [13]:
```
a,b = 0,1                 # Tuple unpacking
count = 0       # initializing new variable
while b < 100:            # to print fibonacci numbers less than 100
    print b,
    a,b = b, a+b
    count+=1   # newVariables can't be used in loops, as new variable gets cre
ated everytime

print "\nIn total, there are %d fibonacci numbers"%(count)
```

```
1 1 2 3 5 8 13 21 34 55 89
In total, there are 11 fibonacci numbers
```

In [153]:

```
#!/usr/bin/python

#class5_whileFor.py


print "\nExample to illustrate CONTINUE "
print "with while loop"
j = 0
while j<10:
  if j == 5:
      j+=1       # comment this line and observe the difference
      continue
  print j,
  j+=1

print "\nwith for loop"
for j in range(10):
  if j==5:
    continue
  print j,

print "\nExample to illustrate BREAK "
print "with while loop"
p = 0
while p<10:
  if p == 5:
      break
  print p,
  p+=1

print "\nwith for loop"
for p in range(10):
  if p==5:
     break
  print p,

print "\nExample to illustrate PASS "
print "with while loop"
i = 0
while i<10:
  if i == 5:
      pass
  print i,
  i+=1

print "\nwith for loop"
for i in range(10):
  if i==5:
     pass       # its like TODO
  print i,

print "\nExample to illustrate sys.exit "

import sys

print "with while loop"
i = 0
```

```
while i<10:
  if i == 5:
      sys.exit(0)  # exit code 0 - everything is correct. 1- some error
  print i,
  i+=1

print "\nwith for loop"
for i in range(10):
  if i==5:
     sys.exit(1)
  print i,
```

```
Example to illustrate CONTINUE
with while loop
0 1 2 3 4 6 7 8 9
with for loop
0 1 2 3 4 6 7 8 9
Example to illustrate BREAK
with while loop
0 1 2 3 4
with for loop
0 1 2 3 4
Example to illustrate PASS
with while loop
0 1 2 3 4 5 6 7 8 9
with for loop
0 1 2 3 4 5 6 7 8 9
Example to illustrate sys.exit
with while loop
0 1 2 3 4

An exception has occurred, use %tb to see the full traceback.

SystemExit: 0

--------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\python27\lib\site-packages\IPython\core\interactiveshell.pyc in run_code(s
elf, code_obj, result)
   2875                result.error_in_exec = e
   2876            self.showtraceback(exception_only=True)
-> 2877            warn("To exit: use 'exit', 'quit', or Ctrl-D.", level=1)
   2878        except self.custom_exceptions:
   2879            etype, value, tb = sys.exc_info()

TypeError: 'level' is an invalid keyword argument for this function
```

Observe that sys.exit() will exit the program execution. Also, observe that last for loop wasn't executed.

In [14]:
```
# Generating the POWER Series:
# e**x =1+x+x**2/2! +x**3/3! +....+ x**n/n! where 0 < x < 1
# class5_powerSeries.py
#!/usr/bin/python
x = float(input("Enter the value of x: "))
n = term = num = 1  # multiple Assigment
sum = 1.0
while n <= 100:
    term *= x / n
    sum += term
    n += 1
    if term < 0.0001:
        break
print("No of Times= %d and Sum= %f" % (n, sum)) # print function
```

```
Enter the value of x: 3
No of Times= 15 and Sum= 20.085523
```

In [16]:
```
#!/usr/bin/python
# class5_multiplicationTable.py
# Write a Script to print the multiplication table up to 10

maxLimit = 14
i = 1
print "-"*20
while i<maxLimit:
    n =1
    while n<maxLimit:
       print '%2d * %2d = %3d'%(i,n,i*n),
       #print '%3d'%(i*n),
       n+=1
       print '|'
       i+=1

print "-"*20
```

```
--------------------
 1 *  1 =   1 |
 2 *  2 =   4 |
 3 *  3 =   9 |
 4 *  4 =  16 |
 5 *  5 =  25 |
 6 *  6 =  36 |
 7 *  7 =  49 |
 8 *  8 =  64 |
 9 *  9 =  81 |
10 * 10 = 100 |
11 * 11 = 121 |
12 * 12 = 144 |
13 * 13 = 169 |
--------------------
```

Assignment 1: Run the below class5_multiplicationTables2.py, with the various commented options, in the script. Observe the differences.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# class5_multiplicationTables2.py

'''
    Purpose:
        To print the multiplications tables upto 10
'''

maxLimit = 10
#maxLimit = int(raw_input('Enter the maximum table to display :'))
print '-'*50
i=1
while i<maxLimit+1:
    j =1
    while j< maxLimit+1:
        #print "i = ", i,"j = " , j, "i*j = ", i*j
        #print i, '*', j, '=', i*j
        #print "%d * %d = %d "%(i,j,i*j)
        print '%2d * %2d = %3d'%(i,j, i*j)#,
        j+=1
    #print i
    print '-'*25
    i+=1

print '-'*50
```

Assignment 2: Based on class5_multiplicationTables2.py, write a program to display the multiplication tables from 1 through 10, horizontally

```
ex:
 1 * 1 = 1 | 2 * 1 = 2 | 3 * 1 = 3 | ....
 1 * 2 = 2 | 2 * 2 = 4 | 3 * 2 = 6 | ....
 ...
```

In [17]:
```
#!/usr/bin/python

# class5_halfDiamond.py

# To display the astrickes in a half-diamond pattern
size = 10
j=0
#print 'j=',j
while j<size:
        print '*'*j
        j+=1

#print 'j=',j

while j>0:
        print '*'*j
        j-=1

#print 'j=',j

# implementation with for loop
for j in range(size):
    print '*'*j

for j in range(size,0, -1):
    print '*'*j
```

```
*
**
***
****
*****
******
*******
********
*********
**********
*********
********
*******
******
*****
****
***
**
*

*
**
***
****
*****
******
*******
********
*********
**********
*********
********
*******
******
*****
****
***
**
*
```

In [18]:
```
#!/usr/bin/python

# class5_quaterdiamond.py

# Printing a quarter diamond
row = int(input("Enter the number of rows: "))
n = row
while n >= 0:
    x = "*" * n
    y = " " * (row - n)
    print(y + x)
    n -= 1

print 'row = %d'%(row)
print 'n = %d'%(n)

n = row
while n >= 0:
    x = "*" * n
    y = " " * (row - n)
    print(x+y)
    n -= 1
```

```
Enter the number of rows: 5
*****
 ****
  ***
   **
    *

row = 5
n = -1
*****
****
***
**
*
```

Assignment 3: Write a program to display a full diamond shape, separetely using for loop, and using while loop?

Collections

List

- List can be classified as single-dimensional and multi-dimensional.
- List is representing using `[]`
- List is a mutable object, which means elements in list can be changed.
- It can store asymmetric data types.

```
In [21]: list1 = [12, 23, 34, 56, 67, 89]   # Homogenous list

In [22]: type(list1)

Out[22]: list

In [23]: l2 = [12, 23.45, 231242314125, 'Python', True, str(0), complex(2,3), 2+3j,
         int(23.45)]

In [24]: l2        # non-homogenous list

Out[24]: [12, 23.45, 231242314125L, 'Python', True, '0', (2+3j), (2+3j), 23]

In [25]: type(l2)

Out[25]: list

In [27]: l2[3], type(l2[3])   # indexing

Out[27]: ('Python', str)

In [28]: l2[6], type(6)       # elements of a list, retain their data type.

Out[28]: ((2+3j), int)
```

In [29]:
```
print dir(l2)        # results in the attributes and methods associated with
 'list' type
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__del
slice__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '_
_getitem__', '__getslice__', '__gt__', '__hash__', '__iadd__', '__imul__', '_
_init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__
new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul_
_', '__setattr__', '__setitem__', '__setslice__', '__sizeof__', '__str__', '_
_subclasshook__', 'append', 'count', 'extend', 'index', 'insert', 'pop', 'rem
ove', 'reverse', 'sort']
```

In [30]:
```
l1 = []             # empty list
```

In [31]:
```
l1, type(l1)
```

Out[31]:
```
([], list)
```

In [32]:
```
l1 = [12, 34]       # Re-initialization. Previous object gets overwritten
```

In [33]:
```
l1.append(23)
```

In [34]:
```
l1
```

Out[34]:
```
[12, 34, 23]
```

In [35]:
```
l1.append(['Python', 3456])
```

In [36]:
```
l1
```

Out[36]:
```
[12, 34, 23, ['Python', 3456]]
```

In [37]:
```
l1.append([56])    # It is different from l1.append(56)
```

In [38]:
```
l1
```

Out[38]:
```
[12, 34, 23, ['Python', 3456], [56]]
```

In [39]:
```
l1.extend(78)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-39-7defae611412> in <module>()
----> 1 l1.extend(78)

TypeError: 'int' object is not iterable
```

In [40]:
```
l1.extend([78])
```

In [41]:
```
l1
```

Out[41]:
```
[12, 34, 23, ['Python', 3456], [56], 78]
```

In [42]:
```
l1.extend(['Django', 45567])
```

In [43]:
```
l1
```

Out[43]:
```
[12, 34, 23, ['Python', 3456], [56], 78, 'Django', 45567]
```

Interview Question 1: In which cases, list.extend is more suitable than list.append?

extend - add the new list to the original list, in the same dimension

append - adds the new list to the original list, in separate dimension

In [44]:
```
print list1+l1
```

```
[12, 23, 34, 56, 67, 89, 12, 34, 23, ['Python', 3456], [56], 78, 'Django', 45
567]
```

In [45]:
```
l2 = [12, [23, 45], [56]]
```

In [47]:
```
l3 = [12, [23, 45], [56]]
```

In [48]:
```
l2+l3
```

Out[48]:
```
[12, [23, 45], [56], 12, [23, 45], [56]]
```

In [49]:
```
l2.append(l3)  # adding in new dimension
```

In [50]:
```
l2
```

Out[50]:
```
[12, [23, 45], [56], [12, [23, 45], [56]]]
```

In [51]:
```
l2 = [12, [23, 45], [56]]  # reassigning
```

In [52]:
```
l2.extend(l3)
```

In [53]:
```
l2   # It is same as l2+l3
```

Out[53]:
```
[12, [23, 45], [56], 12, [23, 45], [56]]
```

In [54]:
```
l2.insert(0, 'Yash')
```

In [55]:
```
l2
```

Out[55]:
```
['Yash', 12, [23, 45], [56], 12, [23, 45], [56]]
```

In [57]:
```
l2.insert(5, 999)   # specifying position is mandatory
```

In [58]:
```
l2
```

Out[58]:
```
['Yash', 12, [23, 45], [56], 12, 999, [23, 45], [56]]
```

NOTE: list.insert() is different from overwritting

In [59]:
```
l2[5] = 'Nine Nine'
```

In [60]:
```
l2
```

Out[60]:
```
['Yash', 12, [23, 45], [56], 12, 'Nine Nine', [23, 45], [56]]
```

In [61]:
```
l2.insert(3,[23, ''])
```

In [62]:
```
l2
```

Out[62]:
```
['Yash', 12, [23, 45], [23, ''], [56], 12, 'Nine Nine', [23, 45], [56]]
```

In [63]:
```
l2 = [12, [23, 45], [56]]  # reassigning
```

In [64]:
```
len(l2)
```

Out[64]:
```
3
```

In [65]:
```
l2.insert(len(l2), l3)
```

In [66]:
```
l2                    # It is same as l2.extend(l3)
```

Out[66]:
```
[12, [23, 45], [56], [12, [23, 45], [56]]]
```

In [67]:
```
l2.insert(45, 34)   # there isn't 45th position. so, added in the last
```

In [68]:
```
l2
```

Out[68]:
```
[12, [23, 45], [56], [12, [23, 45], [56]], 34]
```

In [69]:
```
l2.pop()   # outputs last element, and removes from list
```

Out[69]:
```
34
```

In [70]:
```
l2
```

Out[70]:
```
[12, [23, 45], [56], [12, [23, 45], [56]]]
```

In [73]:
```
l2 = [12, [23, 45], [56]]  # reassigning
```

In [74]:
```
l2.remove(12)   # nothing will be outputted
```

In [75]:
```
l2
```

Out[75]:
```
[[23, 45], [56]]
```

In [76]:
```
l2.remove(56)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-76-33f84afb8304> in <module>()
----> 1 l2.remove(56)

ValueError: list.remove(x): x not in list
```

In [77]:
```
l2.remove([56])
```

In [78]:
```
l2
```

Out[78]:
```
[[23, 45]]
```

In [79]:
```
l2.remove([23])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-79-5a17760cd7ae> in <module>()
----> 1 l2.remove([23])

ValueError: list.remove(x): x not in list
```

In [80]:
```
l2[0], type(l2)
```

Out[80]:
```
([23, 45], list)
```

In [81]:
```
l2[0].remove(23)
```

In [82]:
```
l2
```

Out[82]:
```
[[45]]
```

In [83]:
```
l2 = [12, [23, 45], [56]]  # reassigning
```

In [84]:
```
del l2[1]     # deleting an element in list
```

In [85]:
```
l2
```

Out[85]:
```
[12, [56]]
```

In [86]:
```
del l2        # deleting a list object
```

In [87]:
```
l2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-87-c7ff24afc800> in <module>()
----> 1 l2

NameError: name 'l2' is not defined
```

In [91]:
```
l3 = ['zero', 0, '0', 'Apple', 1, '1']
```

In [92]:
```
sorted(l3)   # builtin function  ; doesn't affect the original object
```

Out[92]:
```
[0, 1, '0', '1', 'Apple', 'zero']
```

In [93]:
```
l3
```

Out[93]:
```
['zero', 0, '0', 'Apple', 1, '1']
```

In [95]:
```
l3.sort()   # modifies the original object
```

In [96]:
```
l3
```

Out[96]:
```
[0, 1, '0', '1', 'Apple', 'zero']
```

In [97]:
```
reversed(l3)
```

Out[97]:
```
<listreverseiterator at 0x4a8ec90>
```

In [98]:
```
for i in reversed(l3):
    print i,
```

```
zero Apple 1 0 1 0
```

In [99]:
```
l4 = []
for i in reversed(l3):
    l4.append(i)

print l4       # This is list type
```

```
['zero', 'Apple', '1', '0', 1, 0]
```

In [100]:
```
l3       # original object didn't change
```

Out[100]:
```
[0, 1, '0', '1', 'Apple', 'zero']
```

In [102]:
```
l3.reverse()
```

In [103]:
```
l3
```

Out[103]:
```
['zero', 'Apple', '1', '0', 1, 0]
```

In [104]:
```
l3.count(1)
```

Out[104]: 1

In [105]:
```
l3.count('1')
```

Out[105]: 1

In [106]:
```
l3.append('1')
```

In [107]:
```
l3
```

Out[107]: ['zero', 'Apple', '1', '0', 1, 0, '1']

In [108]:
```
l3.count('1')
```

Out[108]: 2

In [109]:
```
l3.append(['1'])
```

In [110]:
```
l3
```

Out[110]: ['zero', 'Apple', '1', '0', 1, 0, '1', ['1']]

In [111]:
```
l3.count('1')   # dimension account here
```

Out[111]: 2

In [112]:
```
l3.count(['1'])
```

Out[112]: 1

List Comprehensions

syntax:

```
[logic for i  in (initialValue, finalValue, increment/decrement)  if condition]
```

In [113]:
```
week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

In [114]:
```
type(week), type(week[2])
```

Out[114]: (list, str)

In [115]:
```
print [i for i in week]
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

In [116]:
```
for i in week:
    print i,
```

```
Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

In [117]:
```
[print i for i in week] #stdout not possible within comprehension
```

```
  File "<ipython-input-117-7deeef36a48a>", line 1
    [print i for i in week] #stdout not possible within comprehension
         ^
SyntaxError: invalid syntax
```

In [118]:
```
print [i.upper() for i in week]
```

```
['MONDAY', 'TUESDAY', 'WEDNESDAY', 'THURSDAY', 'FRIDAY', 'SATURDAY', 'SUNDA
Y']
```

In [119]:
```
print [i.lower() for i in week]
```

```
['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunda
y']
```

In [120]:
```
print [i.capitalize() for i in week]
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunda
y']
```

In [121]:
```
print [i[0:3] for i in week]
```

```
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

In [123]:
```
print [i[:3] for i in week]
```

```
['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']
```

In [122]:
```
print [i[0:2] for i in week]
```

```
['Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa', 'Su']
```

In [124]:
```
print [i[3] for i in week]
```

```
['d', 's', 'n', 'r', 'd', 'u', 'd']
```

In [125]:
```
print [i for i in week if i.startswith('T')]
```

```
['Tuesday', 'Thursday']
```

NOTE: else and elif are not possible in list comprehensions

In [127]:
```
print [i for i in week if i.startswith('W')]
```

```
['Wednesday']
```

In [128]:
```
print [i for i in week if len(i) == 8]
```

```
['Thursday', 'Saturday']
```

In [129]:
```
ord('a')  # returns the ACSCII value of the character
```

Out[129]:
```
97
```

In [130]:
```
print [ord(i) for i in 'Python Programming']
```

```
[80, 121, 116, 104, 111, 110, 32, 80, 114, 111, 103, 114, 97, 109, 109, 105,
 110, 103]
```

In [131]:
```
chr(80)  # returns the correpsonding ACSII character for the number
```

Out[131]:
```
'P'
```

In [132]:
```
print [chr(i) for i in range(12)]
```

```
['\x00', '\x01', '\x02', '\x03', '\x04', '\x05', '\x06', '\x07', '\x08',
 '\t', '\n', '\x0b']
```

In [133]:
```
print [chr(i) for i in [80, 121, 116, 104, 111, 110, 32, 80, 114, 111, 103, 11
4, 97, 109, 109, 105, 110, 103]]
```

```
['P', 'y', 't', 'h', 'o', 'n', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm', 'm',
 'i', 'n', 'g']
```

In [134]:
```
''.join([chr(i) for i in [80, 121, 116, 104, 111, 110, 32, 80, 114, 111, 103,
114, 97, 109, 109, 105, 110, 103]])
```

Out[134]:
```
'Python Programming'
```

In [135]:
```
print [float(i) for i in range(9)]
```

```
[0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
```

Assignment 4: Generate a Caesar Cipher for a given string using list comprehensions

In [136]:
```
for i in range(4):
    print i, i*i
```

```
0 0
1 1
2 4
3 9
```

In [137]:
```
print [i, i*i for i in range(9)]
```

```
  File "<ipython-input-137-3c71da0ffd9d>", line 1
    print [i, i*i for i in range(9)]
                      ^
SyntaxError: invalid syntax
```

it will not work, as objects resulting from logic are not qualified

In [138]:
```
print [[i, i*i] for i in range(9)]
```

```
[[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8, 64]]
```

In [139]:
```
print [(i, i*i) for i in range(9)]
```

```
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64)]
```

In [140]:
```
print [[i, i*i, i*i*i] for i in range(9)]
```

```
[[0, 0, 0], [1, 1, 1], [2, 4, 8], [3, 9, 27], [4, 16, 64], [5, 25, 125], [6,
 36, 216], [7, 49, 343], [8, 64, 512]]
```

In [141]:
```
[[i, i*i, i*i*i] for i in range(9)]
```

Out[141]:
```
[[0, 0, 0],
 [1, 1, 1],
 [2, 4, 8],
 [3, 9, 27],
 [4, 16, 64],
 [5, 25, 125],
 [6, 36, 216],
 [7, 49, 343],
 [8, 64, 512]]
```

Nested List comprehensions

In [143]:
```
[(i,i*i, i*i*i, i**4) for i in range(9)]
```

Out[143]:
```
[(0, 0, 0, 0),
 (1, 1, 1, 1),
 (2, 4, 8, 16),
 (3, 9, 27, 81),
 (4, 16, 64, 256),
 (5, 25, 125, 625),
 (6, 36, 216, 1296),
 (7, 49, 343, 2401),
 (8, 64, 512, 4096)]
```

Flattening a multi-dimensional list

In [144]:
```
matrix = [[11, 22, 33],
          [22, 33, 11],
          [33, 11, 22]]
```

In [145]:
```
matrix
```

Out[145]:
```
[[11, 22, 33], [22, 33, 11], [33, 11, 22]]
```

In [146]:
```
flattened = []
for row in matrix:
    for n in row:
        flattened.append(n)
```

In [147]:
```
print flattened
```

```
[11, 22, 33, 22, 33, 11, 33, 11, 22]
```

In [148]:
```
flattenedLc = [n for row in matrix for n in row]
```

In [149]:
```
flattenedLc
```

Out[149]:
```
[11, 22, 33, 22, 33, 11, 33, 11, 22]
```

Assignment 5: Pythogorean triples, between 1 and 55

```
a**2 + b**2 = c**2
```

ex: [3,4,5] WAP to result these triples using for loop, while loop and list comprehension

Composite List Comprehensions

In [150]:
```
['Good' if i else "Bad" for i in range(3)]
```

Out[150]:
```
['Bad', 'Good', 'Good']
```

In [151]:
```
# It the same as the below
for i in range(3):
    if i:
        print "Good"
    else:
        print "Bad"
```

```
Bad
Good
Good
```

In [152]:
```
["Good" if not i else "Bad" for i in range(3)]
```

Out[152]:
```
['Good', 'Bad', 'Bad']
```

Assignment 6: Implement Queue mechanism using lists (FIFO - Queue)

Assignment 5: If the temperature on 22 July, 2015 was recorded as 32°C, 44°C, 45.5°C and 22°C in Paris, Hyderabad, Visakhapatnam and Ontario respectively, what are their temperatures in °F.

```
Hint: T(°F) = T(°C) × 9/5 + 32
```

Assignment 6: If the temperatures on same day this year are forecasted as same numerics in °F, what will be their corresponding temperature in °C.

Interview Question 2: What is the sorting algorithm used in python?

Ans: timesort

Interview Question 3 : What is the difference between sort() and sorted()?

Ans: list.sort() is a nethod of list data structure;whereas sorted(list) is a build-in function. list.sort() will modify the existing list object; sorted(list) will create a new object, without modifying the existing list object. Limitation of list comprehensions is that pass, break, continue won't work in them.