2.0 Python Basics

IDLE will be installed, along with basic python in Windows. In Linux and Unix, it can be installed manually. IDLE is a python IDE, from Python. Python commands can be executed using, either:

```
1. Interactive Mode, or
2. Script Mode
```

Individual commands can be executed in executed in interactive mode. Script mode is preferred for write a program.

In script mode, >>> indicates the prompt of the python interpreter.

```
Programming in Python:
    1. Interactive Mode Programming
            Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500
 32 bit (Intel)] on win32
            Type "help", "copyright", "credits" or "license" for more information.
            >>>
    2. Script Mode Programming
            $ python script.py

            #!/usr/bin/python
            print "Hello, World!"

            $ chmod +x script.py
            $ ./script.py
```

2.1 Basic Syntax and Indenting

In [1]:
```
a = 12
```

In [2]:
```
print a
```

```
12
```

In [3]:
```
b = 34
```

In [4]:
```
print "b = ", b
```

```
b =  34
```

```
>>>  a=12
  File "<stdin>", line 1
    a=12
    ^
IndentationError: unexpected indent
```

```
In [5]: for i in [1,2,335]:
        print i
```

```
  File "<ipython-input-5-4fc7dc342d36>", line 2
    print i
        ^
IndentationError: expected an indented block
```

```
In [6]: for i in [1,2,335]:
            print i
```

```
1
2
335
```

```
In [7]: if True:
                print "Something"
        else:
        print "Nothing"
```

```
  File "<ipython-input-7-e309279ca877>", line 4
    print "Nothing"
        ^
IndentationError: expected an indented block
```

```
In [8]: if True:
                print "Something"
        else:
                print "Nothing"
```

```
Something
```

So, ensure that indentation is provided whenever it is needed, and avoid undesired indendations. Python Program works based on indentation.

PEP 8 is a python group for coding style. It recommends **4 spaces** as indentation. Also, they recommend to prefer spaces, to tabs. If any one is interested in using tabs, ensure that the tab space is configured to 4 spaces, in settings of your editor or IDE.

Also, there should be consistency of intendation, throughtout the program.

2.2 Identifier Naming Conventions

Identifier can represent an object, including variables, classes, functions, execption, ...

```
For Identifiers, first character must be an alphabet (A to Z, a to z) or underscore
 (_)
From second character onwards, it can be alpha-numeric (A to Z, a to z, 0 to 9) and
 underscore (_) character.

Ex: animal, _animal, animal123, ani123mal, ani_mal123, ani12ma_l3 are possible

Ex: 123animal, animal&, $animal, ani$mal, 0animal are not possible. (All these resu
lt in SyntaxError)

And, comma(,), dot(.), % operators are defined in python

    Naming Conventions
        - Class names start with an uppercase letter. All other identifiers start w
ith a lowercase letter.
        - PRIVATE identifiers start with single underscore
           ex: _identierName
        - STRONGLY PRIVATE identifiers start with two leading underscores.
            ex: __identifierName
        - Language defined Special Names - identifier with starts and ends with two
 underscores
            ex: __init__, __main__, __file__
```

Python is **case-sensitive language**. This case-sensitivity can be removed using advanced settings, but it is strongly not recommended.

In [9]:
```
animal = "Cat"
Animal = "Cow"
```

In [10]:
```
print animal
```
```
Cat
```

In [11]:
```
print Animal
```
```
Cow
```

Identifier casing is of two-types:

```
1. snake casing
    ex: cost_of_mangos
2. Camel casing
    ex: costOfMangos
```

PEP 8 recommends to follow any one of them, but, only one type of them in a project.

NOTE: In all the following exercises and examples, Camel casing will be followed.

Comment operator

```
# comment Operator
Interpretor ignore the line, right to this operator
The is only line comment, in python.
```

Docstrings

```
'''  '''

"""  """
```

In [12]:
```
print '''
    These are not multi-line comments, but
    are called docstrings.
    docstrinsg will be processed by the interpreter.
    triple double quotes will also work as docstrings.
'''
```

```
    These are not multi-line comments, but
    are called docstrings.
    docstrinsg will be processed by the interpreter.
    triple double quotes will also work as docstrings.
```

In [13]:
```
'''
    These are not multi-line comments, but
    are called docstrings.
    docstrinsg will be processed by the interpreter.
    triple double quotes will also work as docstrings.
'''
```

Out[13]:
```
'\n    These are not multi-line comments, but\n    are called docstrings.\n
  docstrinsg will be processed by the interpreter.\n    triple double quotes
 will also work as docstrings.\n'
```

Quotes

```
- single ('apple' , "mango"), and triple quotes ('''apple''', """mango""")
- Triple quotes are generally used for docstrings
- Double quotes are NOT allowed. Don't be confused.
- quotes are used in defining strings
    - words, sentences, paragraphs
```

Multi-Line Statements

```
- \ Line continuation operator. (Also, used as reverse division operator)
```

In [14]:
```
SomeOperation = 12+34- 1342342 + 23454545 + 3123 + \
                       3455 - 3454 - 3454 - \
                       234
```

In [15]:
```
print "SomeOperation = ", SomeOperation
```

```
SomeOperation =  22111685
```

Statements used within [], {}, or () doesn't need Line continuation operator

In [16]:
```
months = ('Jan', 'Feb', 'Mar',
                          'Apr', 'May', 'Jun',
                          'jul', 'Aug')
```

In [17]:
```
print months
```

```
('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'jul', 'Aug')
```

Mutiple Statements in a line

```
- ; operator is used to separate statements
```

In [18]:
```
a = 12
b = 34
c = a + b
print "c = ", c
```

```
c =  46
```

```
In [19]: a = 12; b = 34; c = a + b; print "c = ", c      # there are 4 statements in th
         is line
```

```
c =  46
```

2.3 Reserved Keywords in Python:

```
Reserved Keywords  (27 in python 2.x)
--------------------------------------
and       assert       break       class       continue         def
   del
elif      else         except      exec        finally          for
   from
global    if           import      in          is               lambda
   not
or        pass         print       raise       return           try
   while
yield
```

```
Reserved Keywords (33 in python 3.x)
-------------------------------------
False    class       finally      is          return
None     continue    for          lambda      try
True     def         from         nonlocal    while
and      del         global       not         with
as       elif        if           or          yield
assert   else        import       pass
break    except      in           raise
```

These reserved keywords should not be used for the names of user-defined identifiers.

Built-in Functions(64)

`abs()`	`divmod()`	`input()`	`open()`	`staticmethod()`
`all()`	`enumerate()`	`int()`	`ord()`	`str()`
`any()`	`eval()`	`isinstance()`	`pow()`	`sum()`
`basestring()`	`execfile()`	`issubclass()`	`print()`	`super()`
`bin()`	`file()`	`iter()`	`property()`	`tuple()`
`bool()`	`filter()`	`len()`	`range()`	`type()`
`bytearray()`	`float()`	`list()`	`raw_input()`	`unichr()`
`callable()`	`format()`	`locals()`	`reduce()`	`unicode()`
`chr()`	`frozenset()`	`long()`	`reload()`	`vars()`
`classmethod()`	`getattr()`	`map()`	`repr()`	`xrange()`
`cmp()`	`globals()`	`max()`	`reversed()`	`zip()`
`compile()`	`hasattr()`	`memoryview()`	`round()`	`__import__()`
`complex()`	`hash()`	`min()`	`set()`	
`delattr()`	`help()`	`next()`	`setattr()`	
`dict()`	`hex()`	`object()`	`slice()`	
`dir()`	`id()`	`oct()`	`sorted()`	

In [20]:
```
a = 12
print type(a)  # type() returns the type of the object.
```
```
<type 'int'>
```

In [21]:
```
print type(type)
```
```
<type 'type'>
```

In [22]:
```
print id(a)  # returns the address where object 'a' is stored
```
```
2372180
```

In [23]:
```
print(a)   # print() function is different from print statement
```
```
12
```

In [24]:
```
print(dir(a))  # returns the attributes and methods associated with the object
 'a'
```
```
['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__', '__de
lattr__', '__div__', '__divmod__', '__doc__', '__float__', '__floordiv__', '_
_format__', '__getattribute__', '__getnewargs__', '__hash__', '__hex__', '__i
ndex__', '__init__', '__int__', '__invert__', '__long__', '__lshift__', '__mo
d__', '__mul__', '__neg__', '__new__', '__nonzero__', '__oct__', '__or__', '_
_pos__', '__pow__', '__radd__', '__rand__', '__rdiv__', '__rdivmod__', '__red
uce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod_
_', '__rmul__', '__ror__', '__rpow__', '__rrshift__', '__rshift__', '__rsub_
_', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__su
b__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_lengt
h', 'conjugate', 'denominator', 'imag', 'numerator', 'real']
```

In [25]:
```
help(a)   # returns information and usage about the specified object, or funct
ion, ..
```

```
Help on int object:

class int(object)
 |  int(x=0) -> int or long
 |  int(x, base=10) -> int or long
 |  
 |  Convert a number or string to an integer, or return 0 if no arguments
 |  are given.  If x is floating point, the conversion truncates towards zer
o.
 |  If x is outside the integer range, the function returns a long instead.
 |  
 |  If x is not a number or if base is given, then x must be a string or
 |  Unicode object representing an integer literal in the given base.  The
 |  literal can be preceded by '+' or '-' and be surrounded by whitespace.
 |  The base defaults to 10.  Valid bases are 0 and 2-36.  Base 0 means to
 |  interpret the base from the string as an integer literal.
 |  >>> int('0b100', base=0)
 |  4
 |  
 |  Methods defined here:
 |  
 |  __abs__(...)
 |      x.__abs__() <==> abs(x)
 |  
 |  __add__(...)
 |      x.__add__(y) <==> x+y
 |  
 |  __and__(...)
 |      x.__and__(y) <==> x&y
 |  
 |  __cmp__(...)
 |      x.__cmp__(y) <==> cmp(x,y)
 |  
 |  __coerce__(...)
 |      x.__coerce__(y) <==> coerce(x, y)
 |  
 |  __div__(...)
 |      x.__div__(y) <==> x/y
 |  
 |  __divmod__(...)
 |      x.__divmod__(y) <==> divmod(x, y)
 |  
 |  __float__(...)
 |      x.__float__() <==> float(x)
 |  
 |  __floordiv__(...)
 |      x.__floordiv__(y) <==> x//y
 |  
 |  __format__(...)
 |  
 |  __getattribute__(...)
 |      x.__getattribute__('name') <==> x.name
 |  
 |  __getnewargs__(...)
 |  
 |  __hash__(...)
 |      x.__hash__() <==> hash(x)
 |  
 |  __hex__(...)
 |      x.__hex__() <==> hex(x)
 |  
 |  __index__(...)
 |      x[y:z] <==> x[y.__index__():z.__index__()]
 |  
 |  __int__(...)
 |      x.__int__() <==> int(x)
 |  
 |  __invert__(...)
 |      x.__invert__() <==> ~x
 |  
 |  __long__(...)
 |      x.__long__() <==> long(x)
 |  
 |  __lshift__(...)
 |      x.__lshift__(y) <==> x<<y
 |  
 |  __mod__(...)
 |      x.__mod__(y) <==> x%y
 |  
 |  __mul__(...)
 |      x.__mul__(y) <==> x*y
 |  
 |  __neg__(...)
 |      x.__neg__() <==> -x
 |  
 |  __nonzero__(...)
 |      x.__nonzero__() <==> x != 0
 |  
 |  __oct__(...)
 |      x.__oct__() <==> oct(x)
 |  
 |  __or__(...)
 |      x.__or__(y) <==> x|y
 |  
 |  __pos__(...)
 |      x.__pos__() <==> +x
 |  
 |  __pow__(...)
 |      x.__pow__(y[, z]) <==> pow(x, y[, z])
 |  
 |  __radd__(...)
 |      x.__radd__(y) <==> y+x
 |  
 |  __rand__(...)
 |      x.__rand__(y) <==> y&x
 |  
 |  __rdiv__(...)
 |      x.__rdiv__(y) <==> y/x
 |  
 |  __rdivmod__(...)
 |      x.__rdivmod__(y) <==> divmod(y, x)
 |  
 |  __repr__(...)
 |      x.__repr__() <==> repr(x)
 |  
 |  __rfloordiv__(...)
 |      x.__rfloordiv__(y) <==> y//x
 |  
 |  __rlshift__(...)
 |      x.__rlshift__(y) <==> y<<x
 |  
 |  __rmod__(...)
 |      x.__rmod__(y) <==> y%x
 |  
 |  __rmul__(...)
 |      x.__rmul__(y) <==> y*x
 |  
 |  __ror__(...)
 |      x.__ror__(y) <==> y|x
 |  
 |  __rpow__(...)
 |      y.__rpow__(x[, z]) <==> pow(x, y[, z])
 |  
 |  __rrshift__(...)
 |      x.__rrshift__(y) <==> y>>x
 |  
 |  __rshift__(...)
 |      x.__rshift__(y) <==> x>>y
 |  
 |  __rsub__(...)
 |      x.__rsub__(y) <==> y-x
 |  
 |  __rtruediv__(...)
 |      x.__rtruediv__(y) <==> y/x
 |  
 |  __rxor__(...)
 |      x.__rxor__(y) <==> y^x
 |  
 |  __str__(...)
 |      x.__str__() <==> str(x)
 |  
 |  __sub__(...)
 |      x.__sub__(y) <==> x-y
 |  
 |  __truediv__(...)
 |      x.__truediv__(y) <==> x/y
 |  
 |  __trunc__(...)
 |      Truncating an Integral returns itself.
 |  
 |  __xor__(...)
 |      x.__xor__(y) <==> x^y
 |  
 |  bit_length(...)
 |      int.bit_length() -> int
 |      
 |      Number of bits necessary to represent self in binary.
 |      >>> bin(37)
 |      '0b100101'
 |      >>> (37).bit_length()
 |      6
 |
 |  conjugate(...)
 |      Returns self, the complex conjugate of any int.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  denominator
 |      the denominator of a rational number in lowest terms
 |
 |  imag
 |      the imaginary part of a complex number
 |
 |  numerator
 |      the numerator of a rational number in lowest terms
 |
 |  real
 |      the real part of a complex number
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __new__ = <built-in method __new__ of type object>
 |      T.__new__(S, ...) -> a new object with type S, a subtype of T
```

2.4 Arithmetic Operations

Arithmetic Operators:

```
+ - * / \ % ** // =
```

PEP 8 recommends to place one space around the operator

In [26]:
```
var1 = 123       # int
var2 = 2345      # int
```

Addition

In [27]:
```
print var1+var2   # int
```
```
2468
```

In [28]:
```
var3 = 23.45      # float
```

In [29]:
```
print type(var1), type(var2), type(var3)
```
```
<type 'int'> <type 'int'> <type 'float'>
```

```
In [30]: print var1 + var3  # type-casting takes place  # int + float = float
```

```
146.45
```

```
In [31]: var4 = 453453454534545435345435345345345454357090970970707
```

```
In [32]: print var4
```

```
453453454534545435345435345345345454357090970970707
```

```
In [33]: var4    # observe the 'L' in the end
```

```
Out[33]: 453453454534545435345435345345345454357090970970707L
```

```
In [34]: print type(var4)
```

```
<type 'long'>
```

Assignment 1: what is the largest number that can be computed in python?

```
In [35]: print var2 + var4   # int + long int
```

```
453453454534545435345435345345345454357090970973052
```

```
In [36]: print var3 + var4  # float + long int
```

```
4.53453454535e+50
```

```
In [37]: print var2
```

```
2345
```

```
In [38]: var2 = 234.456   # overwrite the existing object; dynamic typing

         print var2
```

```
234.456
```

```
In [39]: print type(var2)
```

```
<type 'float'>
```

subtraction

```
In [40]: print var1 - var2
```

```
-111.456
```

```
In [41]: print var2 - var4
```

```
-4.53453454535e+50
```

Assignment 2: what is the smallest number that can be computed in python?

Multiplication

```
In [42]: print var1 * var2  # int * int
```

```
28838.088
```

```
In [43]: print var1 * var2  # int * float
```

```
28838.088
```

```
In [44]: print var1, var2
```

```
123 234.456
```

```
In [45]: print var3, var4
```

```
23.45 453453454534545435345435345345345454357090970970707
```

```
In [46]: var5 = 23
```

```
In [47]: print var1 * var5 # int * int
```

```
2829
```

```
In [48]: print var1 * var4 # int * long int
```

```
55774774907749088547488547477477490885922189429396961
```

Division Operation

Division is different in python 2.x and python 3.x

```
/   division operator
//  floor division operator
\   reverse division (deprecated). It is no more used.
```

```
In [49]: 10/5
```

```
Out[49]: 2
```

```
In [50]: 10/2

Out[50]: 5

In [51]: 10/3

Out[51]: 3

In [52]: 10//3

Out[52]: 3

In [53]: 10/3.0   # true division in python 2

Out[53]: 3.3333333333333335
```

In python3, 10/3 will give true division

```
In [54]: 2/10

Out[54]: 0
```

\ reverse division operator got deprecated

```
In [55]: 2\10

           File "<ipython-input-55-1bdd425914b1>", line 1
             2\10
                       ^
         SyntaxError: unexpected character after line continuation character

In [56]: 10/3                 # int/int = int

Out[56]: 3

In [57]: 10/3.0               # int/float = float

Out[57]: 3.3333333333333335

In [58]: 10.0/3               # float/int = float

Out[58]: 3.3333333333333335

In [59]: 10.0/3.0             # float/float = float

Out[59]: 3.3333333333333335

In [60]: float(3)   # float() is a built-in function, used to convert to floating-point
          value

Out[60]: 3.0
```

```
In [61]: 10/float(3)
Out[61]: 3.3333333333333335
```

```
In [62]: 2/10
Out[62]: 0
```

```
In [63]: 2.0/10
Out[63]: 0.2
```

```
In [64]: 2.0//10
Out[64]: 0.0
```

```
In [65]: 5/2.0
Out[65]: 2.5
```

```
In [66]: 5//2.0
Out[66]: 2.0
```

```
In [67]: 5//2   #float division will convert to floor(), after division
Out[67]: 2
```

power operation

** - power operator

pow() - builtin function

```
In [68]: 2 ** 3
Out[68]: 8
```

```
In [69]: 3 ** 100
Out[69]: 515377520732011331036461129765621272702107522001L
```

```
In [70]: pow(2,3)
Out[70]: 8
```

```
In [71]: pow(4,0.5)  # square root
Out[71]: 2.0
```

```
In [72]: pow(4,0.652)
Out[72]: 2.469125213787077
```

```
In [73]: 4 ** 0.652
Out[73]: 2.469125213787077
```

```
In [74]: print var1, var2

         123 234.456
```

```
In [75]: var1 ** var2

         ---------------------------------------------------------------------------
         OverflowError                             Traceback (most recent call last)
         <ipython-input-75-c04ac7f60ff9> in <module>()
         ----> 1 var1 ** var2

         OverflowError: (34, 'Result too large')
```

```
In [76]: pow(var1, var2)

         ---------------------------------------------------------------------------
         OverflowError                             Traceback (most recent call last)
         <ipython-input-76-4eb667228a30> in <module>()
         ----> 1 pow(var1, var2)

         OverflowError: (34, 'Result too large')
```

exponent operation

```
In [77]: 1e10
Out[77]: 10000000000.0
```

```
In [78]: 1e1  # equal to 1.0 * 10 **1
Out[78]: 10.0
```

```
In [79]: 1 * 10 **1
Out[79]: 10
```

```
In [80]: 1.0 * 10 **1
Out[80]: 10.0
```

Modulo Operations

```
In [81]: 0%3, 1%3, 2%3, 3%3, 4%3, 5%3, 6%3, 7%3, 8%3, 9%3, 10%3

Out[81]: (0, 1, 2, 0, 1, 2, 0, 1, 2, 0, 1)
```

Observe that there are 3 elements repeating (0, N-1) where N is modulo divisor.

You can take the analogy of an Analog clock. After it completes 12 hours, it starts again from 0

```
In [82]: 0%12, 1%12, 11%12

Out[82]: (0, 1, 11)

In [83]: 12%12

Out[83]: 0

In [84]: 14%12  # 2 past noon

Out[84]: 2

In [85]: 16.45%12            # Observe the output precision

Out[85]: 4.449999999999999

In [86]: -18%12              # Observe the +ve sign in the result

Out[86]: 6

In [87]: -18%-12

Out[87]: -6

In [88]: 18%-12              # conclusion: sign of modulo number is reflected

Out[88]: -6
```

Complex Numbers

Complex Number = Real Number +/- Imaginary Number

In python, 'j' is used to represent the imaginary number.

```
In [89]: print 2 + 3j

         (2+3j)

In [90]: n1 = 2 + 3j
In [91]: print type(n1)
```

```
<type 'complex'>
```

```
In [92]: n2 = 3 - 2j

         print "n2 = ", n2, " type(n2) = ", type(n2)
```

```
n2 =  (3-2j)  type(n2) =  <type 'complex'>
```

```
In [93]: print n1, n1 + n2, n1 - n2
```

```
(2+3j) (5+1j) (-1+5j)
```

```
In [94]: print n1 * n1, pow(n1,2), n1/n1
```

```
(-5+12j) (-5+12j) (1+0j)
```

```
In [95]: print n1, n1.conjugate()    # Observe the signs of imaginary numbers
```

```
(2+3j) (2-3j)
```

```
In [96]: print n1, n1.real, n1.imag
```

```
(2+3j) 2.0 3.0
```

```
In [97]: n2 = 2.0 - 3.45j
```

```
In [98]: print n2, n2.real, n2.imag
```

```
(2-3.45j) 2.0 -3.45
```

```
In [99]: 4j
```

```
Out[99]: 4j
```

NOTE: 4*j, j4, j*4 are not possible. In these cases, interpreter treats 'j' as a variable.

```
In [100]: print n1 + n2, n1 - n2, -n1 + n2, -n1 - n2
```

```
(4-0.45j) 6.45j -6.45j (-4+0.45j)
```

```
In [101]: print n1*n2.real, (n1*n2).real
```

```
(4+6j) 14.35
```

```
In [102]: print n1, n2, n1.real+n2.imag   # n2.imag is resulting in a real value
```

```
(2+3j) (2-3.45j) -1.45
```

```
In [103]: complex(2,-3.456)       # complex()  - Builtin function
Out[103]: (2-3.456j)
```

.* (of Matlab) operator is not valid in python. Element-wise multiplication is possible with numpy module. floor division is also not valid on complex type data.

```
In [104]: (3 + 4j) == (4j + 3)  # == checks value equivalence
Out[104]: True
```

```
In [105]: (3 + 4j) is (4j + 3)  # is - checks object level (both value and address)
Out[105]: False
```

abs()

- Builtin function, to return the absolute value

```
If a is positive real integer,
abs(a)  = a
abs(-a) = a
abs((a+bj))  is equal to math.sqrt(pow(a,2), pow(b,2))
```

```
In [106]: print n1, abs(n1)  # abs - absolute function, Builtin function.
          (2+3j) 3.60555127546
```

```
In [107]: print abs(3)
          3
```

```
In [108]: print abs(-3)
          3
```

```
In [109]: print abs(3 + 4j)
          5.0
```

```
In [110]: import math

          print math.sqrt(3**2+4**2)
          5.0
```

In [111]:
```
divmod(12 + 2j, 2 - 3j) # divmod(x,y) returns x//y, x%y
```
```
c:\python27\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: co
mplex divmod(), // and % are deprecated
  if __name__ == '__main__':
```
Out[111]:
```
((1+0j), (10+5j))
```

Compound(mixed) Operations

+=, -=, *=, **=

In [112]:
```
a = 12
print a, type(a)
```
```
12 <type 'int'>
```

In [113]:
```
a = a + 1 ; print "a = ", a   # ; is used to write multiple statement in same
 line
```
```
a =  13
```

In [114]:
```
a += 1 ; print "a = ", a
```
```
a =  14
```

In [115]:
```
a -= 1 ; print "a = ", a   # a = a -1
```
```
a =  13
```

In [116]:
```
a *= 2 ; print "a = ", a  # a = a*2
```
```
a =  26
```

In [117]:
```
a /= 2 ; print "a = ", a  # a = a / 2
```
```
a =  13
```

In [118]:
```
a **= 2 ; print "a = ", a  # a = a ** 2
```
```
a =  169
```

In [119]:
```
a %= 100 ; print "a = ", a
```
```
a =  69
```

In [120]:
```
a = 23; a//=2; print "a = ", a
```
```
a =  11
```

In [121]:
```
a <<= 1; print "a = ", a   # left-shift
```
```
a =  22
```

```
at binary level 128 64 32 16 8 4 2 1
                             1 0 1 1
            <<1   0  0  0  1 0 1 1 0
```

In [122]:
```
a >>= 1; print "a = ", a   # right-shift
```
```
a =  11
```

```
at binary level 128 64 32 16 8 4 2 1
             11   0  0  0  0 1 0 1 1
              2   0  0  0  0 0 0 1 0
         -----------------------------
              ^   0  0  0  0 1 0 0 1    9
```

In [124]:
```
a^=2; print "a = ", a   # bitwise XOR operation
```
```
a =  9
```

```
at binary level 128 64 32 16 8 4 2 1
              9   0  0  0  0 1 0 0 1
              2   0  0  0  0 0 0 1 0
         -----------------------------
              |   0  0  0  0 1 0 1 1    11
```

In [125]:
```
a|=2; print "a = ", a   # bitwise OR operation
```
```
a =  11
```

NOTE: Pre- and Post- increment/ decrements (++a, a++, --a, a--) are not valid in Python

Working in Script Mode

In [126]:
```
#!/usr/bin/python
# This is called shebong line

# prog1.py

print "Hello World!"
```
```
Hello World!
```

In [127]:
```
#!/usr/bin/python

'''
    DocStrings must come immediately after shebang line.
    These are not multi-line comments, but
    are called docstrings.
    docstrinsg will be processed by the interpreter.
    triple double quotes will also work as docstrings.
'''

# prog2.py

# This hash/pound is the comment operator, used for
# both single line and multi-line comments.
# comment line will be ignored by interpreter

#either single, single or double quotes, can be used for strings


costOfMango = 12
print "cost Of Each Mango is ", costOfMango
costOfApple = 40
print "cost Of Each Apple is ", costOfApple

# what is the cost of dozen apples and two dozens of mangos

TotalCost = 12* costOfApple + 2*12* costOfMango

print "Total cost is ", TotalCost


# print is a statement in python 2, and is a function call in python 3


# now, python 2 is supporting both

print "Hello World!"
print("Hello World!")




# by default, print will lead to display in next line

print "This is",   # , after print will suppress the next line
                   # but, a space will result
print "python class"

# PEP 8 recommends to use only print statement or function call throughtout th
e project

# ; semicolon operator
# It is used as a statement separator.

name = 'yash'
print 'My name is ', name

name = 'yash'; print 'My name is ', name
```

```
print "who's name is ", name, '?'

print "'"
print '"'
print '\''

print "'''  '''"

print '"""'
print "'''"

print ''' """ """ '''
print """ ''' ''' """
```

```
cost Of Each Mango is  12
cost Of Each Apple is  40
Total cost is  768
Hello World!
Hello World!
This is python class
My name is  yash
My name is  yash
who's name is  yash ?
'
"
'
'''  '''
""
''
 """ """
 ''' '''
```

In [128]:
```
#!/usr/bin/python

# prog3.py

# Operator precedence in python
# It follows PEMDAS rule, and left to right, and top to bottom
# P - Paranthesis
# E - Exponent
# M - Multiplication
# D - Division
# A - Addition
# S - Subtraction

#Every type of braces has importance in python
# {}  - used for dictionaries and sets
# []  - used for lists
# ()  - used of tuples; also used in arithmetic operations


result1 = (22+ 2/2*4//4-89)
print "result1 = ", result1

result2 = 32/2/2/2/2
print "result2 = ", result2

result3 = 2 + (3.0 - 5j).imag + 2 ** 3 * 2
print "result3 = ", result3
```

```
result1 =  -66
result2 =  2
result3 =  13.0
```

Assignment 3: Examine the operator precedence will other examples

IO Operations

In python 2.x, raw_input() and input() are two builtin functions used for getting runtime input.

```
raw_input()  - takes any type of runtime input as a string.

input()      - takes any type of runtime input originally without any type conversi
on.


NOTE: working with raw_input() requires us to use type converters to convert the da
ta into the required data type.
```

In Python 3.x, there is only input() function; but not raw_input(). The Job of raw_input() in python 2.x is done by input() in python 3.x

In [129]:
```
#!/usr/bin/python

# class3_io.py

'''
    Purpose : demonstration of input() and raw_input()

'''

dataRI = raw_input('Enter Something: ')

dataI = input('Enter something: ')

print "dataRI = ", dataRI, " type(dataRI) = ", type(dataRI)

print "dataI = ", dataI, " type(dataI) = ", type(dataI)
```

```
Enter Something: 999
Enter something: 999
dataRI =  999  type(dataRI) =  <type 'str'>
dataI =  999  type(dataI) =  <type 'int'>
```

Analyzed outputs for various demonstrated cases:

```
>>> 
==================== RESTART: C:/pyExercises/class3_io.py ====================
Enter Something: 123
Enter something: 123
123 <type 'str'>
123 <type 'int'>
>>> 
==================== RESTART: C:/pyExercises/class3_io.py ====================
Enter Something: 'Yash'
Enter something: 'Yash'
'Yash' <type 'str'>
Yash <type 'str'>
>>> 
==================== RESTART: C:/pyExercises/class3_io.py ====================
Enter Something: True
Enter something: True
True <type 'str'>
True <type 'bool'>
>>> 
==================== RESTART: C:/pyExercises/class3_io.py ====================
Enter Something: Yash
Enter something: Yash

Traceback (most recent call last):
  File "C:/pyExercises/class3_io.py", line 12, in <module>
    dataI = input('Enter something: ')
  File "<string>", line 1, in <module>
NameError: name 'Yash' is not defined
>>> dataRI
'Yash'
```

input() takes only qualified data as runtime input. Whereas **raw_input()** will qualify any data as a 'str' type

In [131]:
```python
#!/usr/bin/python

# class3_io1.py

'''
    Purpose : demonstration of input() and raw_input()

'''

dataRI = int(raw_input('Enter a number: '))

dataI = input('Enter a number: ')


print "dataRI = ", dataRI, " type(dataRI) = ", type(dataRI)

print "dataI = ", dataI, " type(dataI) = ", type(dataI)

print "Sum of numbers is ", dataRI+dataI
```

```
Enter a number: 234
Enter a number: 234
dataRI =  234  type(dataRI) =  <type 'int'>
dataI =  234  type(dataI) =  <type 'int'>
Sum of numbers is  468
```

Analyzed outputs for various demonstrated cases:

```
=================== RESTART: C:/pyExercises/class3_io1.py ===================
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
>>>
=================== RESTART: C:/pyExercises/class3_io1.py ===================
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
Sum of numbers is

Traceback (most recent call last):
  File "C:/pyExercises/class3_io1.py", line 19, in <module>
    print "Sum of numbers is ", dateRI+dataI
NameError: name 'dateRI' is not defined
>>>
=================== RESTART: C:/pyExercises/class3_io1.py ===================
Enter a number: 123
Enter a number: 123
123 <type 'str'>
123 <type 'int'>
Sum of numbers is

Traceback (most recent call last):
  File "C:/pyExercises/class3_io1.py", line 19, in <module>
    print "Sum of numbers is ", dataRI+dataI
TypeError: cannot concatenate 'str' and 'int' objects
>>>
=================== RESTART: C:/pyExercises/class3_io1.py ===================
Enter a number: 123
Enter a number: 123
123 <type 'int'>
123 <type 'int'>
Sum of numbers is  246
>>>
```

INFERENCE:

1. input() takes only qualified objects as inputs; whereas raw_input() considers any input as string data.
2. input() processess the data before taking as input; It is sensed as a security threat by many developers.

String Operations

string data type can be representing using either single or double quotes

Creating a string

```
In [132]: s1 = 'Python Programming'      # single quotes
```

```
In [133]: s1
```

```
Out[133]: 'Python Programming'
```

```
In [134]: print s1
```

```
Python Programming
```

```
In [135]: print type(s1)
```

```
<type 'str'>
```

```
In [136]: s2 = "Django"                  # double quotes
```

```
In [137]: print s2, type(s2)
```

```
Django <type 'str'>
```

```
In [138]: s3 = ''' python programming with Django ''' # triple single quotes
```

```
In [139]: print s3
```

```
 python programming with Django
```

```
In [140]: print type(s3)
```

```
<type 'str'>
```

```
In [141]: s4 = """ python programming with Django  """  # triple double quotes
```

```
In [142]: print s4
```

```
 python programming with Django
```

```
In [143]: type(s4)
```

```
Out[143]: str
```

```
In [144]: print django1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-144-5701398af8ee> in <module>()
----> 1 print django1

NameError: name 'django1' is not defined
```

```
In [145]: print 'django1'
```

```
django1
```

```
In [146]: s5 = '~!@#$%^& *()1232425'  # Any special character can be taken within string
          s
```

```
In [147]: print s5, type(s5)
```

```
~!@#$%^& *()1232425 <type 'str'>
```

```
In [148]: s6 = str(123.34)            # str() is a builtin function to convert to string
```

```
In [149]: print s6, type(s6)
```

```
123.34 <type 'str'>
```

```
In [150]: s7 = str(True)
```

```
In [151]: print s7, type(s7)
```

```
True <type 'str'>
```

Indexing

```
In [152]: print s1
```

```
Python Programming
```

```
In [153]: print len(s1)   # len() is a bultin function to return the length of object
```

```
18
```

```
 P y t h o n      P   r   o  g  r  a  m  m  i  n  g
 0 1 2 3 4 5   6   7   8   9 10 11 12 13 14 15 16 17  -> forward indexing
            -12  -11  -10 -9 -8 -7 -6 -5 -4 -3 -2 -1  -> Reverse indexing
```

```
In [154]: s1[0]

Out[154]: 'P'

In [155]: s1[6]   # white-space character

Out[155]: ' '

In [156]: s1[17]

Out[156]: 'g'

In [157]: s1[18]

          ---------------------------------------------------------------------------
          IndexError                                Traceback (most recent call last)
          <ipython-input-157-2bc6caf9f075> in <module>()
          ----> 1 s1[18]

          IndexError: string index out of range
```

NOTE: Indexing can be done from 0 through len(string)-1

```
In [158]: s1[-1]

Out[158]: 'g'

In [159]: s1[-5]

Out[159]: 'm'

In [160]: s1[-16]

Out[160]: 't'

In [161]: s1[-18] == s1[0]

Out[161]: True
```

Interview Question : what is string[-0]

```
In [162]: s1[-0]  # is equal to s1[0]

Out[162]: 'P'
```

String Slicing

```
In [163]: print s1

          Python Programming
```

```
In [164]: s1[2:6]     # string[InitialBound, finalBound]

Out[164]: 'thon'

In [165]: s1[2:8]

Out[165]: 'thon P'

In [166]: s1[2:17]

Out[166]: 'thon Programmin'

In [167]: s1[2:18]

Out[167]: 'thon Programming'

In [168]: s1[2:19]

Out[168]: 'thon Programming'

In [169]: s1[2:786] # Observe the finalBound

Out[169]: 'thon Programming'

In [170]: s1[2:]  # default finalBound corresponds to lastCharacter in string

Out[170]: 'thon Programming'

In [171]: s1[:]   # defaul initialBound is 0th element

Out[171]: 'Python Programming'

In [172]: s1[:-1]

Out[172]: 'Python Programmin'

In [173]: s1[-5:-1]

Out[173]: 'mmin'

In [174]: s1[-5:17]    # complex indexing

Out[174]: 'mmin'

In [176]: s1[::1]

Out[176]: 'Python Programming'

In [177]: s1[::2]

Out[177]: 'Pto rgamn'
```

```
In [178]: s1[::3]

Out[178]: 'Ph oai'

In [179]: s1[::4]

Out[179]: 'Poran'

In [180]: s1[::-1]

Out[180]: 'gnimmargorP nohtyP'

In [181]: s1[::] # default step is +1

Out[181]: 'Python Programming'

In [182]: s1[:] == s1[::]

Out[182]: True

In [183]: s1[4:9]

Out[183]: 'on Pr'

In [184]: s1[4:9:1]     # string[initialBound, finalBound, increment/decrement]

Out[184]: 'on Pr'

In [185]: s1[4:9:-1]        # 4-1 = 3  index 3 is not represented in this object

Out[185]: ''
```

NOTE: After all these alterations, the original string object will not change, until it is overwrited.

Mutability of Strings

String objects are **immutable**. They, can't be edited. Only way is to overwrite it

```
In [186]: print s1

          Python Programming

In [187]: s1[3]

Out[187]: 'h'
```

```
In [188]: s1[3] = 'H'

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-188-a32a26674f0e> in <module>()
          ----> 1 s1[3] = 'H'

          TypeError: 'str' object does not support item assignment
```

```
In [189]: s1 = "PytHon Programming" # object overwriting taken place

          print s1

          PytHon Programming
```

String attributes

```
In [190]: print dir(s1)

          ['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
           '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
           '__getslice__', '__gt__', '__hash__', '__init__', '__le__', '__len__', '__lt
          __', '__mod__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
          _', '__repr__', '__rmod__', '__rmul__', '__setattr__', '__sizeof__', '__str_
          _', '__subclasshook__', '_formatter_field_name_split', '_formatter_parser',
           'capitalize', 'center', 'count', 'decode', 'encode', 'endswith', 'expandtab
          s', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower', 'i
          sspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partitio
          n', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
           'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translat
          e', 'upper', 'zfill']
```

```
In [192]: s1 = 'PyTHON PROGRAMMING'
          print s1

          PyTHON PROGRAMMING
```

```
In [194]: s1.count('m')

Out[194]: 0
```

```
In [195]: s1.count('M')   # python is case- sensitive, and capitalize() created new obje
          ct,
                          # without disturbing the original object

Out[195]: 2
```

```
In [196]: s1.endswith('ing')  # endswith() returns the boolen result

Out[196]: False
```

```
In [197]: s1.endswith('ING')

Out[197]: True
```

```
In [198]: s1.startswith('Py')

Out[198]: True
```

```
In [199]: s1.find('P')

Out[199]: 0
```

```
In [200]: s1.find('THON')

Out[200]: 2
```

```
In [201]: s1.find('MM')

Out[201]: 13
```

```
In [202]: s1.find('M')

Out[202]: 13
```

```
In [203]: s1.index('THON')

Out[203]: 2
```

Interview Question: Difference between s1.find() and s1. rfind()?

```
In [204]: s1.rfind('P')

Out[204]: 7
```

```
In [205]: s1.rfind('THON')

Out[205]: 2
```

```
In [206]: s1.rfind('MM')

Out[206]: 13
```

```
In [207]: s1.rfind('M')

Out[207]: 14
```

```
In [208]: s1.index('THON')

Out[208]: 2
```

Interview Question: Difference between s1.find() and s1. index()?

```
In [209]: s1.find('Q')

Out[209]: -1
```

```
In [210]: s1.index('Q')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-210-28e3ca7ec558> in <module>()
----> 1 s1.index('Q')

ValueError: substring not found
```

```
In [211]: s1

Out[211]: 'PyTHON PROGRAMMING'
```

```
In [212]: s1.capitalize()

Out[212]: 'Python programming'
```

```
In [213]: s1.lower()

Out[213]: 'python programming'
```

```
In [214]: s1.upper()

Out[214]: 'PYTHON PROGRAMMING'
```

```
In [215]: s1.title()

Out[215]: 'Python Programming'
```

```
In [216]: s1.swapcase()

Out[216]: 'pYthon programming'
```

Interview Question : what is the data type of result of string.split() ?

```
In [217]: s1.split(' ')    # results in 'list' datatype

Out[217]: ['PyTHON', 'PROGRAMMING']
```

```
In [218]: s1.split('O')

Out[218]: ['PyTH', 'N PR', 'GRAMMING']
```

```
In [219]: s1    # Observe that the original object is unchanged

Out[219]: 'PyTHON PROGRAMMING'
```

```
In [220]: s1.split('N')         # string to list conversion

Out[220]: ['PyTHO', ' PROGRAMMI', 'G']

In [221]: s1.split('r')    # no splitting as there is no 'r' character, but 'R' characte
          r in string s1

Out[221]: ['PyTHON PROGRAMMING']

In [222]: s1.split('y')

Out[222]: ['P', 'THON PROGRAMMING']

In [223]: len(s1.split('y'))

Out[223]: 2

In [224]: ''.join(s1.split('y'))    # list to strinsg conversion

Out[224]: 'PTHON PROGRAMMING'

In [225]: '@'.join(s1.split('y'))   # delimiter can be placed

Out[225]: 'P@THON PROGRAMMING'

In [226]: s1.split('O')

Out[226]: ['PyTH', 'N PR', 'GRAMMING']

In [227]: '@'.join(s1.split('O'))   # Observe that 'O' is replaced by '@'.   This is one
           example of duck-typing

Out[227]: 'PyTH@N PR@GRAMMING'

In [228]: s9  = '''
                  This is a good day!
                  Fall 7 times, raise 8!
                  This is a famous japanese quote.
                      '''

In [229]: print len(s9), s9

          114
                  This is a good day!
                  Fall 7 times, raise 8!
                  This is a famous japanese quote.


In [230]: print 'IS'.join(s9.split('is'))

                  ThIS IS a good day!
                  Fall 7 times, raISe 8!
                  ThIS IS a famous japanese quote.
```

In [231]:
```
print 'IS'.join(s9.split(' is'))
```
```
        ThisIS a good day!
        Fall 7 times, raise 8!
        ThisIS a famous japanese quote.
```

In [232]:
```
print ' IS'.join(s9.split(' is'))
```
```
        This IS a good day!
        Fall 7 times, raise 8!
        This IS a famous japanese quote.
```

In [233]:
```
s1
```
Out[233]:
```
'PyTHON PROGRAMMING'
```

In [234]:
```
s1.isalpha()
```
Out[234]:
```
False
```

In [235]:
```
'python'.isalpha()
```
Out[235]:
```
True
```

In [236]:
```
'python programming'.isalpha()  # As there is a space character also
```
Out[236]:
```
False
```

In [237]:
```
'python'.isalnum()
```
Out[237]:
```
True
```

In [238]:
```
'python123'.isalnum()
```
Out[238]:
```
True
```

In [239]:
```
'python123 '.isalnum() # There is a white space character also
```
Out[239]:
```
False
```

In [240]:
```
'python123'.isdigit()
```
Out[240]:
```
False
```

In [241]:
```
'123'.isdigit()
```
Out[241]:
```
True
```

In [242]:
```
'python'.islower()
```
Out[242]:
```
True
```

```
In [243]: 'python123$'.islower()   # It ensures that there is no capital letter, only

Out[243]: True

In [244]: 'python123$ '.isspace()

Out[244]: False

In [245]: ' '.isspace()

Out[245]: True

In [246]: ''.isspace()

Out[246]: False

In [247]: 'python programming'.isupper()

Out[247]: False

In [248]: 'PYTHON'.isupper()

Out[248]: True

In [249]: 'PyTHoN'.isupper()

Out[249]: False

In [250]: s1

Out[250]: 'PyTHON PROGRAMMING'

In [251]: s1.istitle()

Out[251]: False

In [252]: (s1.title()).istitle()

Out[252]: True

In [253]: '   Python  Programming  '.rstrip()

Out[253]: '   Python  Programming'

In [254]: '   Python  Programming  '.lstrip()

Out[254]: 'Python  Programming  '

In [255]: '   Python  Programming  '.strip()   # removes whitespaces

Out[255]: 'Python  Programming'
```

```
In [256]: '   Python  Programming  '.strip('ing')

Out[256]: '   Python  Programming  '

In [257]: '   Python  Programming  '.strip().strip('ing')

Out[257]: 'Python  Programm'

In [258]: 'ad123da'.strip('a')

Out[258]: 'd123d'

In [259]: 'ad1a2a3ada'.strip('a')  # middle characters will retain

Out[259]: 'd1a2a3ad'

In [260]: '01\t012\t0123\t01234'.expandtabs()  # recognizes \t espace character

Out[260]: '01      012     0123    01234'

In [261]: '01012012301234'.expandtabs()

Out[261]: '01012012301234'

In [262]: #To get the website name, excluding the domain
          print 'www.python.org'.lstrip('www.')
          print 'www.python.org'.rstrip('.org')
          print 'www.python.org'.lstrip('www.').rstrip('.org')
          print 'www.python.org'.rstrip('.org').lstrip('www.')
          print 'www.python.org'.strip('www..org')
          print 'www.python.org'.strip('w.org')

          python.org
          www.python
          python
          python
          python
          python

In [263]: 'www.python.org'.split('.')[1]   # alternatively

Out[263]: 'python'

In [264]: 'www.udhayprakash.blogspot.in'.split('.')[1]

Out[264]: 'udhayprakash'

In [265]: word = 'Python'

In [266]: len(word)

Out[266]: 6
```

```
In [267]: word.zfill(6)    # numbers less than len(word) doesn't show any affect

Out[267]: 'Python'

In [268]: word.zfill(7)     # zeros will be prepended correspondingly

Out[268]: '0Python'

In [269]: word.zfill(len(word)+4)

Out[269]: '0000Python'

In [270]: '@'.join((word.zfill(len(word)+4).split('0')))    # filling with character '@'

Out[270]: '@@@@Python'

In [271]: 'Python\tProgramming\t'.expandtabs()   # recognizes '\t', '\r' and '\r\t'; and
           expands correspondingly whenever it finds '\t'

Out[271]: 'Python  Programming     '

In [272]: r'Python\tProgramming'.expandtabs()

Out[272]: 'Python\\tProgramming'

In [273]: 'Python\r\tProgramming'.expandtabs()

Out[273]: 'Python\r        Programming'

In [274]: print 'Python\r\tProgramming'.expandtabs()

                  Programming

In [275]: 'Python\t\rProgramming'.expandtabs()

Out[275]: 'Python  \rProgramming'

In [276]: print 'Python\t\rProgramming'.expandtabs()

          Programming

In [277]: "python programming".partition(' ')

Out[277]: ('python', ' ', 'programming')

In [278]: "python programming".partition('o')

Out[278]: ('pyth', 'o', 'n programming')

In [279]: "python programming".partition('0')   # 0 is not present

Out[279]: ('python programming', '', '')
```

```
In [280]: "python programming".partition('n')
Out[280]: ('pytho', 'n', ' programming')
```

```
In [281]: "python programming".rpartition('n')
Out[281]: ('python programmi', 'n', 'g')
```

```
In [282]: "python programming".partition('g')
Out[282]: ('python pro', 'g', 'ramming')
```

```
In [283]: "python programming".rpartition('n')
Out[283]: ('python programmi', 'n', 'g')
```

```
In [284]: "python programming language".partition('a') # observe 3 'a' characters
Out[284]: ('python progr', 'a', 'mming language')
```

```
In [285]: "python programming language".rpartition('a')  # middle 'a' isn't preferred
Out[285]: ('python programming langu', 'a', 'ge')
```

Assignment : Try to practice the remaining attributes, in this order.

split, splitlines, rsplit

center, ljust, rjust

encode, decode, translate

2.7 String Operations

2.7.6 String Attributes

```
In [286]: myString = 'fall 7 times stand up 8!'
```

```
In [287]: len(myString)
Out[287]: 24
```

```
In [288]: myString.center(len(myString)+1)
Out[288]: ' fall 7 times stand up 8!'
```

```
In [289]: myString.center(len(myString)+2)
Out[289]: ' fall 7 times stand up 8! '
```

```
In [290]: myString.center(len(myString)+5)

Out[290]: '   fall 7 times stand up 8!  '

In [291]: myString.ljust(len(myString)+5)

Out[291]: 'fall 7 times stand up 8!     '

In [292]: myString.rjust(len(myString)+5)

Out[292]: '     fall 7 times stand up 8!'

In [293]: myString.ljust(len(myString)+5).rjust(len(myString)+5)

Out[293]: 'fall 7 times stand up 8!     '

In [294]: myParagraph = "fall 7 times stand up 8!\nfall 7 times stand up 8!\nfall 7 time
          s stand up 8!"

In [295]: print myParagraph

          fall 7 times stand up 8!
          fall 7 times stand up 8!
          fall 7 times stand up 8!

In [296]: myParagraph.splitlines()

Out[296]: ['fall 7 times stand up 8!',
           'fall 7 times stand up 8!',
           'fall 7 times stand up 8!']

In [297]: myParagraph.splitlines(2)   # results in two \n 's

Out[297]: ['fall 7 times stand up 8!\n',
           'fall 7 times stand up 8!\n',
           'fall 7 times stand up 8!']

In [298]: myString.encode('base64')

Out[298]: 'ZmFsbCA3IHRpbWVzIHN0YW5kIHVwIDgh\n'

In [299]: 'ZmFsbCA3IHRpbWVzIHN0YW5kIHVwIDgh\n'.decode('base64')

Out[299]: 'fall 7 times stand up 8!'

In [300]: myString.encode('base64', 'strict')  # 'strict' option results in raising  Uni
          codeError, for encoding errors

Out[300]: 'ZmFsbCA3IHRpbWVzIHN0YW5kIHVwIDgh\n'

In [301]: myString.encode('utf_8', 'strict')

Out[301]: 'fall 7 times stand up 8!'
```

```
In [302]: unicode('fall 7 times stand up 8!')

Out[302]: u'fall 7 times stand up 8!'
```

```
In [303]: type('fall 7 times stand up 8!'), type(unicode('fall 7 times stand up 8!'))

Out[303]: (str, unicode)
```

Interview Question : what is the difference betweem str.find() and in operator

```
In [304]: "c" in "abc"   # in - Membership check operator; returns the boolean(True/Fals
          e)

Out[304]: True
```

```
In [305]: "abc".find("c")

Out[305]: 2
```

```
In [306]: "abc".find("d")

Out[306]: -1
```

```
In [307]: "d" in "abc"

Out[307]: False
```

```
In [308]: "d" not in "abc"

Out[308]: True
```

2.7.8 String Concatenation

```
In [309]: myString

Out[309]: 'fall 7 times stand up 8!'
```

```
In [310]: myNewString = "It's time to wake up!!"
```

```
In [311]: myString + myNewString

Out[311]: "fall 7 times stand up 8!It's time to wake up!!"
```

```
In [312]: ''.join([myString, myNewString])

Out[312]: "fall 7 times stand up 8!It's time to wake up!!"
```

```
In [313]: myString + "@@@" + myNewString

Out[313]: "fall 7 times stand up 8!@@@It's time to wake up!!"
```

```
In [314]: '@@@'.join([myString, myNewString])

Out[314]: "fall 7 times stand up 8!@@@It's time to wake up!!"
```

```
In [315]: myString + 1947

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-315-5a193782e9b7> in <module>()
----> 1 myString + 1947

TypeError: cannot concatenate 'str' and 'int' objects
```

NOTE: python is strictly-typed language.

```
In [316]: myString + str(1947)

Out[316]: 'fall 7 times stand up 8!1947'
```

```
In [317]: str(True) + myString

Out[317]: 'Truefall 7 times stand up 8!'
```

```
In [318]: result = myString + str(True) + str(' ') + str(123.45) + str(' ~!@#$%^&*')
          print type(result), result

<type 'str'> fall 7 times stand up 8!True 123.45 ~!@#$%^&*
```

2.7.9 String Formating

Accessing Arguments by position

```
In [319]: '{}, {} and {}'.format('a', 'b', 'c')

Out[319]: 'a, b and c'
```

```
In [320]: '{}, {} and {}'.format('b', 'b', 'c')

Out[320]: 'b, b and c'
```

```
In [321]: '{0}, {1} and {2}'.format('a', 'b', 'c')

Out[321]: 'a, b and c'
```

```
In [322]: '{0}, {2} and {2}'.format('a', 'b', 'c')

Out[322]: 'a, c and c'
```

```
In [323]: str1 = 'Ram'
          str2 = 'Rahim'
          str3 = 'Robert'

          print "The class is organized by {0}, and attended by {1} and
          {2}".format(str1, str2, str3)

          The class is organized by Ram, and attended by Rahim and Robert
```

```
In [324]: print "The class is organized by {2}, and attended by {0} and
          {1}".format(str1, str2, str3)

          The class is organized by Robert, and attended by Ram and Rahim
```

```
In [325]: '{2}, {1}, {0}'.format(*'abc')   # Tuple unpacking

Out[325]: 'c, b, a'
```

```
In [326]: '{0}, {2}, {1}'.format(*'abc')

Out[326]: 'a, c, b'
```

Accessing arguments by name

```
In [327]: 'Coordinates: {latitude}, {longitude}'.format(latitude='37.24N', longitude='-1
          15.81W')

Out[327]: 'Coordinates: 37.24N, -115.81W'
```

```
In [328]: print "The class is organized by {organizer}, and attended by {std1} and {std
          2}"\
                  .format(organizer = str3, std2 = str2, std1 = str1)

          The class is organized by Robert, and attended by Ram and Rahim
```

2.7.10 str() vs repr() | %s vs %r

Interview Question : what is the difference between str() and repr() ?

```
In [329]: str9 = "C:\newFolder"
```

```
In [330]: print str9

          C:
          ewFolder
```

```
In [331]: str10 = "C:\\newFolder"   # to escape the \ character
In [332]: print str10
```

```
C:\newFolder
```

```
In [333]: str11 = r"C:\newFolder"   #raw string representation
```

```
In [334]: print str11
```

```
C:\newFolder
```

```
In [335]: type(str9), type(str10), type(str11)
```

```
Out[335]: (str, str, str)
```

repr() is representational, but results in string data type.

Interview Question : In which cases, repr() is preferred to str()?

str() is comfortable for Humans; whereas repr() is for the machine

```
In [336]: "repr() shows quotes: {!r}; str() doesn't : {!s}".format('Shoban', 'Shoban')
```

```
Out[336]: "repr() shows quotes: 'Shoban'; str() doesn't : Shoban"
```

```
In [337]: str("udhay")
```

```
Out[337]: 'udhay'
```

```
In [338]: repr("udhay")
```

```
Out[338]: "'udhay'"
```

```
In [339]: str('udhay')
```

```
Out[339]: 'udhay'
```

```
In [340]: repr('udhay')
```

```
Out[340]: "'udhay'"
```

```
In [341]: print str("udhay"), repr("udhay"), str('udhay'), repr('udhay')
```

```
udhay 'udhay' udhay 'udhay'
```

```
In [342]: name = repr("Udhay")
          print "My name is ", name
```

```
My name is  'Udhay'
```

aligning the text and specifying a width

```
In [343]: '{:<30}'.format('Python Programming')  # ljust

Out[343]: 'Python Programming            '
```

```
In [344]: print '{:<30}'.format('Python Programming')  # ljust

          Python Programming
```

```
In [345]: '{:>30}'.format('Python Programming')   # rjust

Out[345]: '            Python Programming'
```

```
In [346]: '{:^30}'.format('Python Programming') # center

Out[346]: '      Python Programming      '
```

Assignment: Create a raw template for the supermarket bill, with corresponding data types. Use string formatting to display the final bill.

Hint: extend previous supermarket logic.

2.8 Logical Operations

```
In [347]: a = 12;
```

```
In [348]: a > 9

Out[348]: True
```

```
In [349]: a < 14

Out[349]: True
```

```
In [350]: a == 34

Out[350]: False
```

```
In [351]: a != 23

Out[351]: True
```

```
In [352]: a <> 23    # <> is also used as != ; used only in python 2.x

Out[352]: True
```

```
In [353]: (a > 9) and (a < 34)

Out[353]: True
```

```
In [354]: (a > 9) and (a > 34)
Out[354]: False
```

```
In [355]: (a > 9) or (a > 34)
Out[355]: True
```

```
In [356]: (a < 9) or (a > 34)
Out[356]: False
```

```
In [357]: not ((a < 9) or (a > 34))
Out[357]: True
```

```
In [358]: (a < 9), not (a < 9)
Out[358]: (False, True)
```

2.9 Bitwise operations

```
In [359]: 4 << 1       # bitwise left-shift - shifts their corresponding binary digits l
          eft side by one position.
                                 # binary notation 8421
Out[359]: 8
```

```
In [360]: 4>>1       # bitwise right-shift
Out[360]: 2
```

```
In [361]: 4 & 8      # bitwise AND    # 4 - 0100        8 - 1000
Out[361]: 0
```

```
In [362]: 4 & 12
Out[362]: 4
```

```
In [363]: 4 | 8      # bitwise OR
Out[363]: 12
```

```
In [364]: 4, ~4      # bitwise not - Gives the 1's complement value
Out[364]: (4, -5)
```

2.10 Identity Operations

- is, is not

```
In [365]: 4 is 4    # is - does object level (value, type, address) identity check
```

```
Out[365]: True
```

```
In [366]: a = 12
          b = 234
```

```
In [367]: a is b
```

```
Out[367]: False
```

```
In [368]: b = 12
```

```
In [369]: a is b
```

```
Out[369]: True
```

```
In [370]: a not is b
```

```
  File "<ipython-input-370-d94cceae31d8>", line 1
    a not is b
           ^
SyntaxError: invalid syntax
```

```
In [371]: a is not b
```

```
Out[371]: False
```

Interview Question : why 'is' operator results in False for two objects with same value above 257, and True for values less than or equal to 256?

```
In [372]: a = 257
          b = 257
```

```
In [373]: a is b
```

```
Out[373]: False
```

```
In [374]: id(a), id(b)   # Observe the addressed of these objects
```

```
Out[374]: (55699820, 55699784)
```

```
In [375]: a = 256
          b = 256
```

```
In [376]: a is b
Out[376]: True
```

```
In [377]: id(a), id(b)             # 1 byte - 256
Out[377]: (2375204, 2375204)
```

NOTE: This is due to the dual-memory management strategy of Python. Till 256, one strategy is applied; whereas from 257, another memory management strategy is applied.

Assignment : Try this with float values, and try to find the boundary?

2.10 Boolean Operations

- True, False

```
In [378]: True
Out[378]: True
```

```
In [379]: False
Out[379]: False
```

```
In [380]: true

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-380-74d9a83219ca> in <module>()
          ----> 1 true

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

```
In [381]: not True
Out[381]: False
```

```
In [382]: True is True
Out[382]: True
```

```
In [383]: print True is False
          False
```

```
In [384]: False is True
Out[384]: False
```

```
In [385]: False is False
Out[385]: True

In [386]: False is not True
Out[386]: True

In [387]: True is not False
Out[387]: True

In [388]: True * 3             # Value for 'True' boolean object is 1
Out[388]: 3

In [389]: False * 9           # Value for 'False' boolean object is 0
Out[389]: 0

In [390]: True == 1
Out[390]: True

In [391]: True is 1
Out[391]: False

In [392]: id(True), id(1)
Out[392]: (505422916, 2372312)

In [393]: True is not 1
Out[393]: True

In [394]: 0 == False
Out[394]: True

In [395]: 0 is False
Out[395]: False

In [396]: bool(23 > 45)    # bool() - builtin function; results in boolean value
Out[396]: False

In [397]: bool('')   # empty or null elements result in False
Out[397]: False
```

```
In [398]: bool('python')

Out[398]: True
```

```
In [399]: bool(0), bool(1), bool(23420), bool(-3424)

Out[399]: (False, True, True, True)
```

```
In [400]: bool([]), bool({}), bool(())

Out[400]: (False, False, False)
```

```
In [401]: bool([12]), bool({12}), bool((12))

Out[401]: (True, True, True)
```

```
In [402]: bool([2,3]), bool({23, 34}), bool((122, 345))

Out[402]: (True, True, True)
```

```
In [403]: bool(True), bool(False), bool(not False)

Out[403]: (True, False, True)
```

2.10 Order of evaluation

```
operators                             descriptions
--------------------------------------------------------------
(), [], {}, ‘’                        tuple, list, dictionnary, string
x.attr, x[], x[i:j], f()              attribute, index, slide, function call
+x, -x, ~x                            unary negation, bitwise invert
**                                    exponent
*, /, %                               multiplication, division, modulo
+, -                                  addition, substraction
<<, >>                                bitwise shifts
&                                     bitwise and
^                                     bitwise xor
|                                     bitwise or
<, <=, >=, >,==, !=                   comparison operators
is, is not, in,not in                 comparison operators (continue)
not                                   boolean NOT
and                                   boolean AND
or                                    boolean OR
lambda                                lambda expression
```

Assignment 3 : Try few arithmetic expressions and Observe that the Order of precedence is followed.

2.11 Conditional Operations

PEP 8 recommends 4 spaces for indentation

In [404]:
```
if True:
    print "By Default, It will be True"
```

```
By Default, It will be True
```

In [405]:
```
if True:
    print "By Default, It will be True"
else:
    print "Unless given as False, It isn't choosen"
```

```
By Default, It will be True
```

In [406]:
```
if 34 < 56:
    print '34<56'
```

```
34<56
```

In [407]:
```
if 1:
    print 'It is true'
else:
    print 'It is false'
```

```
It is true
```

In [408]:
```
if -67:
    print 'It is true'
else:
    print 'It is false'
```

```
It is true
```

In [409]:
```
a = False

if a == True:
    print 'It is true'
else:
    print 'It is false'
```

```
It is false
```

In [410]:
```
if a:                                   # equivalent to a == True
    print 'It is true'
else:
    print 'It is false'
```

```
It is false
```

```
In [411]: if not a:
              print 'It is true'
          else:
              print 'It is false'

          It is true
```

```
In [412]: a = -23
          if a:
              print 'It is not zero'
          else:
              print 'It is zero'

          It is not zero
```

2.11 range() and xrange()

```
In [413]: range(9)

Out[413]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [414]: var = range(9)
          print type(var), var

          <type 'list'> [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [415]: range(0,9)

Out[415]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [416]: range(0,9,1)   # range(initialValue, FinalBound, Step)

Out[416]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
range(initialValue, FinalBound, Step)

defaults:
initialValue = 0th value
FinalBound = last value in list/string
step = +1
```

```
In [417]: range(-1)

Out[417]: []
```

```
In [418]: range(0,9,1)

Out[418]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [419]: range(0,9,2)

Out[419]: [0, 2, 4, 6, 8]
```

```
In [420]: range(0,9,-2)  # 0-2 = -2 , which is out of bounds of [0-9]

Out[420]: []
```

```
In [421]: range(9,0,-2)

Out[421]: [9, 7, 5, 3, 1]
```

```
In [422]: range(9.5,0.5,-0.5)

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-422-08513d47a46c> in <module>()
          ----> 1 range(9.5,0.5,-0.5)

          TypeError: range() integer end argument expected, got float.
```

```
In [423]: print range(-44, -4, 4)

          [-44, -40, -36, -32, -28, -24, -20, -16, -12, -8]
```

```
In [424]: print range(-4, -44, -4)

          [-4, -8, -12, -16, -20, -24, -28, -32, -36, -40]
```

```
In [425]: range(9,-1)   # Here, finalBoundValue = -1; step = +1

Out[425]: []
```

```
In [426]: range(9,1)  # Here, finalBoundValue = 1; step = +1

Out[426]: []
```

Interview Question : What is the difference between range() and xrange()?

range() will result in a list data type. It will store the resulting data in the buffer. For larger values, it will cost more buffer (heap) memory to store all those values. It is memory inefficient.

```
In [427]: print range(34)

          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 2
          1, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]
```

```
In [428]: print xrange(34)

          xrange(34)
```

In [429]:
```
for i in range(34):
    print i,                # , is a newline suppressor
```
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33
```

In [430]:
```
for i in xrange(34):
    print i,
```
```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
 29 30 31 32 33
```

NOTE: When iterating, both xrange() and range() will result the same end output.

In [432]:
```
#!/usr/bin/python

# lotteryTicketGame.py

ticketNumber = raw_input("Enter the ticket Number:")

lotteryNumber = "12AE8Jy4"

if ticketNumber == lotteryNumber:
    print "YOU ARE THE WINNER"
else:
    print "Sorry! But, Try again. You may win very next moment"
```
```
Enter the ticket Number:12ae8jy4
Sorry! But, Try again. You may win very next moment
```

Assignment : create a lottery ticket game, which takes only numbers; then prompt whether the given ticket number is greater, lesser or equal to the lotter ticket number.

3.1 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 [433]: for i in [1,2,3,5]:   # here, initialization is 1,
              print i
```

```
1
2
3
5
```

```
In [434]: 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 [435]: for i in range(6):      # initialization = 0, increment = 1; finalValue = 6
              print i,
```

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

```
In [436]: for i in 'Python':
              print i,
```

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

```
In [437]: 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 [438]: for i in words:
              if i == 'Python':
                  continue
              print i
```

```
Programming
Django
NEtworking
```

In [439]:
```
#!/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.

3.2 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 [440]:
```
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 [441]: 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 [442]: # 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 [443]: 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 [444]: count

Out[444]: 11
```

```
In [445]: 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 [446]:

```
#!/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

c:\python27\lib\site-packages\IPython\core\interactiveshell.py:2889: UserWarn
ing: To exit: use 'exit', 'quit', or Ctrl-D.
  warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)
```

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

In [447]:
```
#!/usr/bin/python
'''
Generating the POWER Series:
 e**x =1+x+x**2/2! +x**3/3! +....+ x**n/n! where 0 < x < 1
 class5_powerSeries.py
'''
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: 4
 No of Times= 18 and Sum= 54.598136
```

In [448]:

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

maxLimit = 14
i = 1
print "-" * 16
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 "-" * 16
```

```
----------------
 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 : 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 : 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 [449]:
```
#!/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 [450]: #!/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: 4
****
 ***
  **
   *

row = 4
n = -1
****
***
**
*
```

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

4 Collections

- Lists
- Tuples
  - named Tuples
- Dictionaries
- sets
- Comprehensions

4.1 Lists

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

```
In [452]: type(list1)

Out[452]: list
```

```
In [453]: myList = [12, 23.45, 231242314125, 'Python', True, str(0), complex(2,3), 2+3j,
           int(23.45)]
```

```
In [454]: print myList        # non-homogenous list

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

```
In [455]: print type(myList)

          <type 'list'>
```

```
In [456]: print myList[3], type(myList[3])   # indexing

          Python <type 'str'>
```

```
In [457]: myList[6], type(myList[6])      # elements of a list, retain their data type.

Out[457]: ((2+3j), complex)
```

```
In [458]: print dir(myList)       # results in the attributes and methods associated wi
          th '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 [459]: myList = []             # empty list
```

```
In [460]: print myList, type(myList)

          [] <type 'list'>
```

```
In [461]: myList = [12, 34]      # Re-initialization. Previous object gets overwritten
```

```
In [462]: print myList.append(23) # Observe that append() operation doesn't return anyth
          ing

          None
```

```
In [463]: print myList

          [12, 34, 23]
```

```
In [464]: myList.append(['Python', 3456])
```

```
In [465]: print myList

          [12, 34, 23, ['Python', 3456]]
```

```
In [466]: myList.append([56])    # It is different from myList.append(56)
```

```
In [467]: print "myList = ", myList

          myList =  [12, 34, 23, ['Python', 3456], [56]]
```

```
In [468]: myList.extend(78)

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-468-0f3f11492ca4> in <module>()
          ----> 1 myList.extend(78)

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

```
In [469]: myList.extend([78])
```

```
In [470]: print myList

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

```
In [471]: myList.extend(['Django', 45567])
```

```
In [472]: print myList

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

Interview Question : 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 [473]: print list1
          print myList
          print list1 + myList

          [12, 23, 34, 56, 67, 89]
          [12, 34, 23, ['Python', 3456], [56], 78, 'Django', 45567]
          [12, 23, 34, 56, 67, 89, 12, 34, 23, ['Python', 3456], [56], 78, 'Django', 45
          567]

In [474]: myList = [12, [23, 45], [56]]

In [475]: myNewList = [12, [23, 45], [56]]

In [476]: myList + myNewList

Out[476]: [12, [23, 45], [56], 12, [23, 45], [56]]

In [477]: myList.append(myNewList)  # adding in new dimension

In [478]: print myList

          [12, [23, 45], [56], [12, [23, 45], [56]]]

In [479]: myList = [12, [23, 45], [56]]  # reassigning

In [480]: myList.extend(myNewList)

In [481]: print myList   # It is same as myList + myNewList

          [12, [23, 45], [56], 12, [23, 45], [56]]

In [482]: myList.insert(0, 'Yash')

In [483]: print myList

          ['Yash', 12, [23, 45], [56], 12, [23, 45], [56]]

In [484]: myList.insert(5, 999)   # specifying position is mandatory

In [485]: myList

Out[485]: ['Yash', 12, [23, 45], [56], 12, 999, [23, 45], [56]]
```

NOTE: list.insert() is different from overwritting

```
In [486]: myList[5] = 'Nine Nine'

In [487]: print myList

          ['Yash', 12, [23, 45], [56], 12, 'Nine Nine', [23, 45], [56]]
```

```
In [488]: myList.insert(3,[23, ''])
```

```
In [489]: print myList

          ['Yash', 12, [23, 45], [23, ''], [56], 12, 'Nine Nine', [23, 45], [56]]
```

```
In [490]: myList = [12, [23, 45], [56]]  # reassigning
```

```
In [491]: print len(myList)

          3
```

```
In [492]: myList.insert(len(myList), myNewList)
```

```
In [493]: myList                    # It is same as myList.extend(myNewList)

Out[493]: [12, [23, 45], [56], [12, [23, 45], [56]]]
```

```
In [494]: myList.insert(45, 34)   # there isn't 45th position. so, added in the last
```

```
In [495]: print myList

          [12, [23, 45], [56], [12, [23, 45], [56]], 34]
```

```
In [496]: print myList.pop()   # outputs last element, and removes from list

          34
```

```
In [497]: print myList

          [12, [23, 45], [56], [12, [23, 45], [56]]]
```

```
In [498]: myList = [12, [23, 45], [56]]  # reassigning
```

```
In [499]: myList.remove(12)   # nothing will be outputted
```

```
In [500]: print myList

          [[23, 45], [56]]
```

```
In [501]: myList.remove(56)

          ---------------------------------------------------------------------------
          ValueError                                Traceback (most recent call last)
          <ipython-input-501-06795c785ca9> in <module>()
          ----> 1 myList.remove(56)

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

```
In [502]: myList.remove([56])
In [503]: print myList

          [[23, 45]]

In [504]: myList.remove([23])

          ---------------------------------------------------------------------------
          ValueError                                Traceback (most recent call last)
          <ipython-input-504-1bcbe61bb7e6> in <module>()
          ----> 1 myList.remove([23])

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

In [505]: myList[0], type(myList)

Out[505]: ([23, 45], list)

In [506]: myList[0].remove(23)

In [507]: print myList

          [[45]]

In [508]: myList = [12, [23, 45], [56]]  # reassigning

In [509]: del myList[1]     # deleting an element in list

In [510]: myList

Out[510]: [12, [56]]

In [511]: del myList        # deleting a list object

In [512]: print myList

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-512-0f7d6aa1a5f7> in <module>()
          ----> 1 print myList

          NameError: name 'myList' is not defined

In [513]: myList3 = ['zero', 0, '0', 'Apple', 1, '1']

In [514]: sorted(myList3)   # builtin function  ; doesn't affect the original object

Out[514]: [0, 1, '0', '1', 'Apple', 'zero']

In [515]: print myList3

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

```
In [516]: myList3.sort()   # modifies the original object

In [517]: print myList3

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

In [518]: reversed(myList3)

Out[518]: <listreverseiterator at 0x3d56cf0>

In [519]: for i in reversed(myList3):
              print i,

          zero Apple 1 0 1 0

In [520]: list(reversed(myList3))

Out[520]: ['zero', 'Apple', '1', '0', 1, 0]

In [521]: myList4 = []
          for i in reversed(myList3):
              myList4.append(i)

          print myList4       # This is list type

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

In [522]: myList3       # original object didn't change

Out[522]: [0, 1, '0', '1', 'Apple', 'zero']

In [523]: myList3.reverse()

In [524]: myList3

Out[524]: ['zero', 'Apple', '1', '0', 1, 0]

In [525]: myList3.count(1)

Out[525]: 1

In [526]: myList3.count('1')

Out[526]: 1

In [527]: myList3.append('1')

In [528]: myList3

Out[528]: ['zero', 'Apple', '1', '0', 1, 0, '1']
```

```
In [529]: myList3.count('1')
Out[529]: 2
```

```
In [530]: myList3.append(['1'])
```

```
In [531]: myList3
Out[531]: ['zero', 'Apple', '1', '0', 1, 0, '1', ['1']]
```

```
In [532]: myList3.count('1')   # dimension account here
Out[532]: 2
```

```
In [533]: myList3.count(['1'])
Out[533]: 1
```

4.1a List Comprehensions

syntax:

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

```
In [534]: week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'S
          unday']
```

```
In [535]: type(week), type(week[2])
Out[535]: (list, str)
```

```
In [536]: print [i for i in week]

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

```
In [537]: for i in week:
              print i,

          Monday Tuesday Wednesday Thursday Friday Saturday Sunday
```

```
In [538]: [print i for i in week]

            File "<ipython-input-538-be84efc03df9>", line 1
              [print i for i in week]
                   ^
          SyntaxError: invalid syntax
```

NOTE: stdout not possible within comprehension

```
In [539]: print [i.upper() for i in week]

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

```
In [540]: print [i.lower() for i in week]

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

```
In [541]: print [i.capitalize() for i in week]

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

```
In [542]: print [i[0:3] for i in week]

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

```
In [543]: print [i[:3] for i in week]

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

```
In [544]: print [i[0:2] for i in week]

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

```
In [545]: print [i[3] for i in week]

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

```
In [546]: print [i for i in week if i.startswith('T')]

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

NOTE: else and elif are not possible in list comprehensions

```
In [547]: print [i for i in week if i.startswith('W')]

          ['Wednesday']
```

```
In [548]: print [i for i in week if len(i) == 8]

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

```
In [549]: ord('a')  # returns the ACSCII value of the character

Out[549]: 97
```

```
In [550]: 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 [551]: chr(80)  # returns the correpsonding ACSII character for the number
Out[551]: 'P'
```

```
In [552]: print [chr(i) for i in range(12)]
```

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

```
In [553]: 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 [554]: ''.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[554]: 'Python Programming'
```

```
In [555]: 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 : Generate a Caesar Cipher for a given string using list comprehensions

```
In [556]: for i in range(4):
              print i, i*i
```

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

4.1b Nested List comprehensions

```
In [557]: print [i, i*i for i in range(9)]
```

```
  File "<ipython-input-557-3c71da0ffd9d>", 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 [558]: 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 [559]: 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 [560]: 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 [561]: [[i, i*i, i*i*i] for i in range(9)]

Out[561]: [[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 [562]: [(i,i*i, i*i*i, i**4) for i in range(9)]

Out[562]: [(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 [563]: matrix = [[11, 22, 33],
                    [22, 33, 11],
                    [33, 11, 22]]

          print len(matrix)
          print matrix
          print len(matrix[0])
          print matrix[0]

          3
          [[11, 22, 33], [22, 33, 11], [33, 11, 22]]
          3
          [11, 22, 33]
```

In [564]:
```
flattened = []
for row in matrix:
    for n in row:
        flattened.append(n)

print flattened
```

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

In [565]:
```
flattenedLc = [n for row in matrix for n in row]

print flattenedLc
```

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

Assignment : 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

4.1c Composite List Comprehensions

In [566]:
```
'Good' if (7 < 9) else "Bad"  # Conditional (ternary) Operation
```

Out[566]:
```
'Good'
```

In [567]:
```
['Good' if i else "Bad" for i in range(3)]  # Duck-tying
```

Out[567]:
```
['Bad', 'Good', 'Good']
```

In [568]:
```
# It the same as the below
for i in range(3):
    if i:
        print "Good"
    else:
        print "Bad"
```

```
Bad
Good
Good
```

In [569]:
```
["Good" if not i else "Bad" for i in range(3)]
```

Out[569]:
```
['Good', 'Bad', 'Bad']
```

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

Assignment : 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 : 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 : What is the sorting algorithm used in python?

Ans: timesort

Interview Question : 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.

Interview Question 5: Implement the stack mechanism using 'List'

Creating a Stack with Lists

- stack works based on Last-In First-Out (LIFO) mechanism.
- The push and pop operations of stack can be mimicked using append() and pop() methods of list, respectively.

```
In [570]: stack = [12, 34, 45, 5677]
```

```
In [571]: print type(stack)

          <type 'list'>
```

```
In [572]: stack.append(25)        # push operation
```

```
In [573]: print stack

          [12, 34, 45, 5677, 25]
```

```
In [574]: stack.pop()             # pop operation

Out[574]: 25
```

```
In [575]: stack.pop()               # LIFO

Out[575]: 5677
```

```
In [576]: stack.append(8880)
```

```
In [577]: print stack

          [12, 34, 45, 8880]
```

Creataing a Queue with Lists

Interview Question : Implement a queue mechanism, using collections module

```
In [578]: from collections import deque
          queue = deque(['Python', 'Programming', 'Pearl'])

          print queue

          deque(['Python', 'Programming', 'Pearl'])
```

```
In [579]: type(queue)      # returns a named tuple # It is of 'collections.deque' type.
           Different from basic col types

Out[579]: collections.deque
```

```
In [580]: print dir(queue)

          ['__class__', '__copy__', '__delattr__', '__delitem__', '__doc__', '__eq__',
           '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash
          __', '__iadd__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne
          __', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '_
          _setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'app
          end', 'appendleft', 'clear', 'count', 'extend', 'extendleft', 'maxlen', 'po
          p', 'popleft', 'remove', 'reverse', 'rotate']
```

```
In [581]: queue.appendleft('George')
```

```
In [582]: queue.appendleft('Bush')
```

```
In [583]: queue

Out[583]: deque(['Bush', 'George', 'Python', 'Programming', 'Pearl'])
```

```
In [584]: queue.pop()

Out[584]: 'Pearl'
```

```
In [585]: queue.pop()

Out[585]: 'Programming'
```

```
In [586]: queue
Out[586]: deque(['Bush', 'George', 'Python'])
```

```
In [587]: queue.clear()    # clears the queue; but retains the queue object
```

```
In [588]: queue
Out[588]: deque([])
```

Interview Question : what is the difference between = and ==

> = assignment operation ; Also called as Hard COPY
>
> == equivalence checking operation

```
In [589]: a = 23  # assigning 23 to 'a'
```

```
In [590]: print a
          23
```

```
In [591]: a == 23
Out[591]: True
```

4.1.2 Hard COPY vs Shallow COPY vs Deep COPY (https://www.youtube.com/watch?v=yjYlyydmrc0&index=2&list=PLTEjme3I6BCg6Pd-KkMaysdtuEG1cAv_0)

Hard COPY is the assignment Operation

```
In [592]: parList = [1,2,3,4,54,5,56,6]
```

```
In [593]: childList = parList  # Assignment Operation (or)  Hard COPY
```

Assignment operation wont create a new object; rather the new identifier (variable) refers to the same Object

```
In [594]: print parList, type(parList)
          [1, 2, 3, 4, 54, 5, 56, 6] <type 'list'>
```

```
In [595]: print childList, type(childList)
          [1, 2, 3, 4, 54, 5, 56, 6] <type 'list'>
```

```
In [596]: parList == childList
Out[596]: True

In [597]: parList is childList    # becoz both are refering to the same object.
Out[597]: True

In [598]: print id(parList), id(childList)

          64414624 64414624

In [599]: parList[4]
Out[599]: 54

In [600]: parList[4] = 'Five Four'

In [601]: parList
Out[601]: [1, 2, 3, 4, 'Five Four', 5, 56, 6]

In [602]: childList       # modifications are reflected in childList
Out[602]: [1, 2, 3, 4, 'Five Four', 5, 56, 6]

In [603]: childList[5] = 'Five'

In [604]: childList
Out[604]: [1, 2, 3, 4, 'Five Four', 'Five', 56, 6]

In [605]: parList         # modifications are reflected in parList
Out[605]: [1, 2, 3, 4, 'Five Four', 'Five', 56, 6]
```

```
import copy

copy.copy()       ->   Shallow COPY
copy.deepcopy()   ->   Deep COPY
```

```
In [606]: parList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Programming']]
           # re-assigning

In [607]: hardCopyList = parList   # Hard COPY or assignment operation
In [608]: import copy

          shallowCopyList = copy.copy(parList)    # shallow COPY
          deepCopyList = copy.deepcopy(parList)   # Deep COPY
```

```
In [609]: print 'parList = %r \nhardCopyList = %r \nshallowCopyList = %r \ndeepCopyList
           = %r'%(\
                   parList, hardCopyList, shallowCopyList, deepCopyList)

          parList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Programming']]
          hardCopyList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Programmin
          g']]
          shallowCopyList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]
          deepCopyList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Programmin
          g']]
```

```
In [610]: print 'id(parList) = %16r \nid(hardCopyList) = %11r \nid(shallowCopyList) = %r
           \nid(deepCopyList) = %11r'%(\
                      id(parList), id(hardCopyList), id(shallowCopyList), id(deepCopyLis
          t))

          id(parList) =         64415424
          id(hardCopyList) =    64415424
          id(shallowCopyList) = 64359400
          id(deepCopyList) =    64415144
```

With this, we can draw inference that shallowCopyList and deepCopyList are creating a new objects

```
In [611]: parList == hardCopyList == shallowCopyList == deepCopyList

Out[611]: True

In [612]: parList is hardCopyList is shallowCopyList is deepCopyList

Out[612]: False

In [613]: parList is hardCopyList

Out[613]: True

In [614]: parList is shallowCopyList

Out[614]: False

In [615]: parList is deepCopyList

Out[615]: False

In [616]: shallowCopyList is deepCopyList

Out[616]: False
```

```
In [617]: parList

Out[617]: [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Programming']]

In [618]: parList[4] = False

In [619]: print 'parList = %75r \nhardCopyList = %70r \nshallowCopyList = %65r \ndeepCop
          yList = %69r'%(\
                     parList, hardCopyList, shallowCopyList, deepCopyList)

          parList =         [12, 23.34, '1223', 'Python', False, [12, 23, '34', 'Progra
          mming']]
          hardCopyList =    [12, 23.34, '1223', 'Python', False, [12, 23, '34', 'Progra
          mming']]
          shallowCopyList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]
          deepCopyList =    [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]

In [620]: hardCopyList[2] = 'NEW STRING'

In [621]: print 'parList = %81r \nhardCopyList = %76r \nshallowCopyList = %65r \ndeepCop
          yList = %69r'%(\
                     parList, hardCopyList, shallowCopyList, deepCopyList)

          parList =         [12, 23.34, 'NEW STRING', 'Python', False, [12, 23, '34',
           'Programming']]
          hardCopyList =    [12, 23.34, 'NEW STRING', 'Python', False, [12, 23, '34',
           'Programming']]
          shallowCopyList = [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]
          deepCopyList =    [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]

In [622]: parList[5][1]

Out[622]: 23

In [623]: parList[5][1] = 88      # changing in second dimension

In [624]: print 'parList = %81r \nhardCopyList = %76r \nshallowCopyList = %65r \ndeepCop
          yList = %69r'%(\
                     parList, hardCopyList, shallowCopyList, deepCopyList)

          parList =         [12, 23.34, 'NEW STRING', 'Python', False, [12, 88, '34',
           'Programming']]
          hardCopyList =    [12, 23.34, 'NEW STRING', 'Python', False, [12, 88, '34',
           'Programming']]
          shallowCopyList = [12, 23.34, '1223', 'Python', True, [12, 88, '34', 'Program
          ming']]
          deepCopyList =    [12, 23.34, '1223', 'Python', True, [12, 23, '34', 'Program
          ming']]
```

Now, let us try in 3rd dimension

```
In [625]: parList = [12, '1223', [23, '34', 'Programming', [56, 45.56, '98.45',
          'Flask']]]
```

```
In [626]: deepCopyList = copy.deepcopy(parList)
```

```
In [627]: parList[2][3][3]
```

```
Out[627]: 'Flask'
```

```
In [628]: parList[2][3][3] = 'Django'
```

```
In [629]: print parList, '\n', deepCopyList
```

```
[12, '1223', [23, '34', 'Programming', [56, 45.56, '98.45', 'Django']]]
[12, '1223', [23, '34', 'Programming', [56, 45.56, '98.45', 'Flask']]]
```

Conclusions:

- In **single dimension** lists, if you do not want the copied list to get affected to the changes in source list, go for **shallow COPY**.
- In **multi-dimensional** lists, if you do not want the copied list to get affected to changes in source list, go for **deep COPY**.

Other Built-in functions

all() Verifies whether all the elements in the collection(list,tuple,set,dictionary) are True or False

any() Verifies whether any of the elements in the collection(list,tuple,set,dictionary) are True or False

```
In [631]: myList = [1, 2, -4, [34, 556, [56, 67, 0]]]
```

```
In [632]: any(myList)
```

```
Out[632]: True
```

```
In [633]: all(myList)    # doesn't consider deep dimensions; only considers the first di
          mension elements
```

```
Out[633]: True
```

```
In [634]: myList.append('')
```

```
In [635]: myList
```

```
Out[635]: [1, 2, -4, [34, 556, [56, 67, 0]], '']
```

```
In [636]: all(myList)        # ''  has False as boolean result
Out[636]: False
```

4.2 Tuples

- Tuples are immutable (means can't be edited)
- Tuples have all the capabilities of lists, expect modification.
- Tuples can be indexed

```
In [637]: t = ()   # empty tuple
```

```
In [638]: print t, type(t)
```

```
() <type 'tuple'>
```

```
In [639]: print dir(t)
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__doc__', '__eq__',
 '__format__', '__ge__', '__getattribute__', '__getitem__', '__getnewargs__',
 '__getslice__', '__gt__', '__hash__', '__init__', '__iter__', '__le__', '__l
en__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
_', '__repr__', '__rmul__', '__setattr__', '__sizeof__', '__str__', '__subcla
sshook__', 'count', 'index']
```

```
In [640]: t2 = ('Apple', 'Mango', 'Goa')       # simple Homogeneous tuple
```

```
In [641]: t3 = ('Apple', 123, 34.56, True)     # simple non-homogenous tuple
```

Buit-in functions can be applied on them

```
In [642]: any(t3)
Out[642]: True
```

```
In [643]: all(t3)
Out[643]: True
```

Indexing and slicing Tuples

```
In [644]: print t2
```

```
('Apple', 'Mango', 'Goa')
```

```
In [645]: print t2[1]
```

```
Mango
```

```
In [646]: t3[:2]

Out[646]: ('Apple', 123)

In [647]: t3[-1]

Out[647]: True

In [648]: t3[1:]

Out[648]: (123, 34.56, True)

In [649]: t3.count(True)

Out[649]: 1

In [650]: t3.index(True)

Out[650]: 3

In [651]: t3

Out[651]: ('Apple', 123, 34.56, True)

In [652]: t3[:]

Out[652]: ('Apple', 123, 34.56, True)

In [653]: t3[::]

Out[653]: ('Apple', 123, 34.56, True)

In [654]: t3[::-1]

Out[654]: (True, 34.56, 123, 'Apple')

In [655]: t3 is t3[:] is t3[::]

Out[655]: True

In [656]: t3 is t3[::-1]

Out[656]: False

In [657]: numbers = range(9)
          print numbers
          print type(numbers)

          [0, 1, 2, 3, 4, 5, 6, 7, 8]
          <type 'list'>
```

```
In [658]: # List to tuple conversion
          tuple(numbers)      # tuple()  - built-in function for converting to tuple

Out[658]: (0, 1, 2, 3, 4, 5, 6, 7, 8)
```

```
In [659]: print type(tuple(numbers)), type(numbers)

          <type 'tuple'> <type 'list'>
```

```
In [660]: t2 + t3   # concatenation

Out[660]: ('Apple', 'Mango', 'Goa', 'Apple', 123, 34.56, True)
```

```
In [661]: t2 + (3,4)

Out[661]: ('Apple', 'Mango', 'Goa', 3, 4)
```

```
In [662]: t2 + (3)

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-662-bf92910fe75d> in <module>()
          ----> 1 t2 + (3)

          TypeError: can only concatenate tuple (not "int") to tuple
```

```
In [663]: t2 + (3,)

Out[663]: ('Apple', 'Mango', 'Goa', 3)
```

```
In [664]: t2 + t3 != t3 + t2   # commutative Property not satisfied

Out[664]: True
```

```
In [665]: t2 * 2  # Repition

Out[665]: ('Apple', 'Mango', 'Goa', 'Apple', 'Mango', 'Goa')
```

```
In [666]: print len(t2), len(t2 * 2)

          3 6
```

```
In [667]: t5 = t2[0:len(t2)-1]    # same as t2[0:2]
          print t5

          ('Apple', 'Mango')
```

in

- Membership verification operator. Used on lists, tuples, strings, dictionaries

```
In [668]: 'Apple' in t5
Out[668]: True
```

```
In [669]: 'Banana' in t5
Out[669]: False
```

```
In [670]: 'Apple ' in t5
Out[670]: False
```

```
In [671]: 'Apples' not in t5
Out[671]: True
```

```
is   -- is not
in   -- not in
```

Assigment : Try all the operations performed on Lists, to tuples, and observe the difference

Tuples are Immutable

```
In [672]: t3
Out[672]: ('Apple', 123, 34.56, True)
```

```
In [673]: t3[0] = 'Kiwi'

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-673-69c0cea87f59> in <module>()
----> 1 t3[0] = 'Kiwi'

TypeError: 'tuple' object does not support item assignment
```

```
In [674]: t2 ** 2   # power operation

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-674-acc8c0d75267> in <module>()
----> 1 t2 ** 2   # power operation

TypeError: unsupported operand type(s) for ** or pow(): 'tuple' and 'int'
```

NOTE: Tuple are Immutable; So, they can't be edited. But, can be overwritten

Buit-in functions on Tuples

```
In [675]: t1 = tuple(range(9)); t2 = tuple(range(3,12))

          print "t1 = ", t1,'\n', "t2 = ", t2

          t1 =  (0, 1, 2, 3, 4, 5, 6, 7, 8)
          t2 =  (3, 4, 5, 6, 7, 8, 9, 10, 11)
```

```
In [676]: t3 = tuple(xrange(9))
          print "t3 = ", t3

          t3 =  (0, 1, 2, 3, 4, 5, 6, 7, 8)
```

```
cmp(obj1, obj2) - builtin function. result in
                            +1 if obj1 is greater,
                            -1 if obj1 is smaller, and
                             0 if both obj1 and obj2 are equal
```

```
In [677]: cmp(t1,t2)

Out[677]: -1
```

```
In [678]: cmp(t2,t1)

Out[678]: 1
```

```
In [679]: cmp(t1,t1)

Out[679]: 0
```

```
In [680]: min(t2)

Out[680]: 3
```

```
In [681]: min((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1)))  # even multidimensional elements are c
          onsidered

Out[681]: -1
```

```
In [682]: min((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1), (-1, 9)))  # even multidimensional eleme
          nts are considered

Out[682]: -1
```

```
In [683]: min((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1, -9)))  # even multidimensional elements a
          re considered

Out[683]: 0
```

Assignment : Try the min() function for lists, and lists within tuples

```
In [684]: max(t2)
Out[684]: 11
```

```
In [685]: max((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1, 100)))  # even multidimensional elements
           are considered
Out[685]: (-1, 100)
```

```
In [686]: max((0, 1, 2, 3, 4, 5, 6, 7, 8, (100), (78,)))
Out[686]: (78,)
```

Inference: (100) is an integer, whereas (78,) is a tuple object.

```
In [687]: max((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1, 100), (100), (-1, -2, 100)))
Out[687]: (-1, 100)
```

```
In [688]: max((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1, 100), [-1, 100] ))
Out[688]: (-1, 100)
```

```
In [689]: max((0, 1, 2, 3, 4, 5, 6, 7, 8, (-1, 100), [-1, 100], {-1,100}, {-1:100} ))
Out[689]: (-1, 100)
```

```
In [690]:  () > set() > [] > {}   # order in collections
Out[690]: True
```

```
In [691]: list(t2)     # list()  - builtin function; used to convert to list type
Out[691]: [3, 4, 5, 6, 7, 8, 9, 10, 11]
```

```
In [692]: sorted(t2)   # convert any collection type, to list; then sorts; Creates new o
          bject
Out[692]: [3, 4, 5, 6, 7, 8, 9, 10, 11]
```

Tuple Unpacking

```
In [693]: a = 12
```

```
In [694]: print a
          12
```

In [695]:
```
a,b  = 420, 950

print "a =", a
print "b =", b

print type(a)
```
```
a = 420
b = 950
<type 'int'>
```

In [696]:
```
a  = 420, 950

print "a =", a


print type(a)
```
```
a = (420, 950)
<type 'tuple'>
```

In [697]:
```
a,b  = 420

print "a =", a
print "b =", b
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-697-4dee3c1eb5cb> in <module>()
----> 1 a,b  = 420
      2 
      3 print "a =", a
      4 print "b =", b

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

NOTE: For unpacking, ensure that the number of identifiers in left side of assignment operator, must be equal to the number of objects(values) on the right-hand side

In [698]:
```
(a,b,c,d,e) = 12, 23, 34, 45, 45

print a, type(a)
```
```
12 <type 'int'>
```

In [699]:
```
(a,b,c,d,e) = (12, 23, 34, 45, 45)

print a, type(a)
```
```
12 <type 'int'>
```

```
In [700]: (a,b,c,d,e) = [12, 23, 34, 45, 45]

          print a, type(a)

          12 <type 'int'>
```

```
In [701]: [a,b,c,d,e] = [12, 23, 34, 45, 45]     # List unpacking

          print a, type(a)

          12 <type 'int'>
```

```
In [702]: print [a,b,c,d,e], type([a,b,c,d,e])

          [12, 23, 34, 45, 45] <type 'list'>
```

Lists within Tuples, and tuples within Lists

```
In [703]: th = (12, 23, 34, [54, 54, 65,(23, 45), [34, 45]], ('python', 'programming'))
```

```
In [704]: len(th)

Out[704]: 5
```

```
In [705]: th[4]

Out[705]: ('python', 'programming')
```

```
In [706]: print type(th[3]), type(th[4])

          <type 'list'> <type 'tuple'>
```

```
In [707]: th[3][0]

Out[707]: 54
```

Interview Question : Can the list elements present within the tuple, be changed?

```
In [708]: th[3][0] = 'Five Four'
```

```
In [709]: print th

          (12, 23, 34, ['Five Four', 54, 65, (23, 45), [34, 45]], ('python', 'programmi
          ng'))
```

```
In [710]: th[4][0]

Out[710]: 'python'
```

```
In [711]: th[4][0] = 'Django'

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-711-1b45a8ec8717> in <module>()
          ----> 1 th[4][0] = 'Django'

          TypeError: 'tuple' object does not support item assignment
```

```
In [714]: th[3][4][0]

Out[714]: 34
```

```
In [715]: th[3][4][0] = 'Three Four'
```

```
In [716]: th[3][3][0]

Out[716]: 23
```

```
In [717]: th[3][3][0] = 'two three'

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-717-f35586ec7407> in <module>()
          ----> 1 th[3][3][0] = 'two three'

          TypeError: 'tuple' object does not support item assignment
```

Inference: The mutability of an element depends on its primary collection type

Assignment : Write a Program to convert this heterogeneous tuple completely to flat tuple.

```
Input = (2,23, 34, [55, 'six six', (77, 88, ['nine nine', 0])])

Output = (2, 23, 34, 55, 'six six', 77, 88, 'nine nine', 0)
```

Tuple Comprehensions (or) Generator expressions

```
In [718]: tc = (i for i in range(9))
```

```
In [719]: print tc

          <generator object <genexpr> at 0x03D7B878>
```

```
In [720]: len(tc)

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-720-a4c5085a07fe> in <module>()
          ----> 1 len(tc)

          TypeError: object of type 'generator' has no len()
```

```
In [721]: print tc.next()

          0
```

```
In [722]: print tc.next()

          1
```

```
In [723]: print tc.next()

          2
```

```
In [724]: print tc.next(), tc.next(), tc.next(), tc.next(), tc.next(), tc.next()

          3 4 5 6 7 8
```

```
In [725]: print tc.next()

          ---------------------------------------------------------------------------
          StopIteration                             Traceback (most recent call last)
          <ipython-input-725-b701980f9cc2> in <module>()
          ----> 1 print tc.next()

          StopIteration:
```

NOTE: Calling tc.next() when there is no value in that, results in StopIteration exception

```
In [726]: tc = (i for i in range(9))
```

```
In [727]: for ele in tc:
              print ele,

          0 1 2 3 4 5 6 7 8
```

```
In [728]: [r for r in tc] # Once it results all the data, there will not be any more ele
          ment in that object

Out[728]: []
```

```
In [729]: tc = (i for i in range(9))
In [730]: [r for r in tc]
```
```
Out[730]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```
```
In [731]: tc = (i for i in range(9))
          tl = [i for i in tc]

          print type(tc), type(tl)
```
```
<type 'generator'> <type 'list'>
```
```
In [732]: [i.next() for i in tc]    # no exception, as there is no element in 'tc'
```
```
Out[732]: []
```

Interview Question : what is the result of this operation:

tc = (i for i in range(9)); [i.next() for i in tc]

```
In [733]: tc = (i for i in range(9))
          [i.next() for i in tc]      # 'i' is an integer, but not an iterator; tc is i
          terator;
                                      # During iteration, elements become basic data ty
          pes
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-733-e300884eaf16> in <module>()
      1 tc = (i for i in range(9))
----> 2 [i.next() for i in tc]      # 'i' is an integer, but not an iterato
r; tc is iterator;
      3                             # During iteration, elements become basi
c data types

AttributeError: 'int' object has no attribute 'next'
```
```
In [734]: [type(i) for i in tc]
```
```
Out[734]: [int, int, int, int, int, int, int, int]
```
```
In [735]: tc = (i for i in [(12, 34), 12, 23, 'String', 'Python', True, 23.2])
```
```
In [736]: [i for i in tc]
```
```
Out[736]: [(12, 34), 12, 23, 'String', 'Python', True, 23.2]
```
```
In [737]: print [type(i) for i in tc]
```
```
[]
```

In [738]:
```
tc = (i for i in [(12, 34), 12, 23, 'String', 'Python', True, 23.2])
print [type(i) for i in tc]
```
```
[<type 'tuple'>, <type 'int'>, <type 'int'>, <type 'str'>, <type 'str'>, <typ
e 'bool'>, <type 'float'>]
```

Assignment : practice all the exercises performed on list comprehensions, on tuple comprehensions

4.3 Sets

- sets are unordered; Then can't be indexed
- sets doesn't store duplicates; It will discard the two and other consequent occurrences of the same element.
- denoted with {}

In [739]:
```
s1 = set()    # empty set   # set() is a builtin function
```

In [740]:
```
print s1, type(s1)
```
```
set([]) <type 'set'>
```

In [741]:
```
s2 = {9,1,2,2,3,4,5,3,5}    # simple set

print s2, type(s2)
```
```
set([1, 2, 3, 4, 5, 9]) <type 'set'>
```

In [742]:
```
s3 = {1,2,[1,2], (1,2), 1,2}   # compound set with a list and a tuple as eleme
nts
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-742-1fd6a6c319ec> in <module>()
----> 1 s3 = {1,2,[1,2], (1,2), 1,2}   # compound set with a list and a tuple
 as elements

TypeError: unhashable type: 'list'
```

NOTE: Elements in a set must be immutable only.

```
mutable  - list
Immutable - tuple, string, int, float, long int
```

In [743]:
```
s3 = {1,2,tuple([1,2]), (1,2), 1,2}

print s3, type(s3)
```
```
set([(1, 2), 1, 2]) <type 'set'>
```

Interview Question : What is the simplest way to remove duplicates in a list?

Ans list(set(list1))

```
In [744]: myList = [45,1,2,3,2,3,3,4,5,6,4,4,4,45,45]
          myList = list(set(myList))   # set() and list() are buit-in functions

          print type(myList), myList

          <type 'list'> [1, 2, 3, 4, 5, 6, 45]
```

```
In [745]: s4  = {'Apple', 'Mango', 'Banana', 12, 'Bnana', 'Mango'}

          print type(s4), s4

          <type 'set'> set([12, 'Mango', 'Bnana', 'Banana', 'Apple'])
```

```
In [746]: print dir(s4)  # set attributes

          ['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__',
           '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
           '__iand__', '__init__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le
          __', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduc
          e__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__seta
          ttr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'a
          dd', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersec
          tion', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop',
           'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'u
          pdate']
```

NOTE: As sets can't be indexed, we need not bother about the way the sequence in which the elements are taken into the set.

NOTE: Sets doesn't support arithmetic Operations.

```
In [747]: s5 = {'Mercedes', 'Toyota', 'Maruthi', 'Hyundai'}
```

```
In [748]: s4 + s5     # + is not even used as concatenation operator on sets

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-748-b57dac89ce6c> in <module>()
          ----> 1 s4 + s5     # + is not even used as concatenation operator on sets

          TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

```
In [749]: s4 - s5  # It means to get elements of s4, which are not present in s5

Out[749]: {12, 'Apple', 'Banana', 'Bnana', 'Mango'}
```

```
In [750]: s5 = {'Mercedes', 'Toyota', 'Maruthi', 'Hyundai', 'Mango'}
In [751]: s4 - s5

Out[751]: {12, 'Apple', 'Banana', 'Bnana'}
```

```
In [752]: # list to tuple conversion

          countries = set(['India', 'Afganistan', 'Sri Lanka', 'Nepal'])

          print type(countries), countries

          <type 'set'> set(['Afganistan', 'Sri Lanka', 'Nepal', 'India'])
```

```
In [753]: # tuple to list conversion

          brics = set(('Brazil', 'Russia', 'India', 'China', 'South Africa'))

          print type(brics), brics

          <type 'set'> set(['Brazil', 'China', 'India', 'Russia', 'South Africa'])
```

Interview Question : what is the result of set('Python Programming')

```
In [754]: # string to set of characters

          strSet = set('Python Programming')

          print type(strSet), strSet

          <type 'set'> set(['a', ' ', 'g', 'i', 'h', 'm', 'o', 'n', 'P', 'r', 't',
           'y'])
```

```
In [755]: # List to set conversion

          strSet1 = set(['Python Programming'])

          print type(strSet1), strSet1

          <type 'set'> set(['Python Programming'])
```

```
In [756]: # tuple to set conversion

          strSet2 = set(('Python Programming'))

          print type(strSet2), strSet2

          <type 'set'> set(['a', ' ', 'g', 'i', 'h', 'm', 'o', 'n', 'P', 'r', 't',
           'y'])
```

Interview Question : what is the result of set(1221)?

```
In [757]: set(1221)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-757-a4823664953d> in <module>()
----> 1 set(1221)

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

```
In [758]: set('1221')
```

```
Out[758]: {'1', '2'}
```

```
In [759]: set([1221])
```

```
Out[759]: {1221}
```

```
In [760]: set([1221, 12221, 1221])
```

```
Out[760]: {1221, 12221}
```

```
In [761]: import sets
```

```
c:\python27\lib\site-packages\ipykernel\__main__.py:1: DeprecationWarning: th
e sets module is deprecated
  if __name__ == '__main__':
```

```
In [762]: # list to sets

asean = sets.Set(['Myanmar', 'Indonesia', 'Malaysia', 'Philiphines', 'Thailan
d'])

print type(asean), asean   # Observe the type of the object
```

```
<class 'sets.Set'> Set(['Malaysia', 'Philiphines', 'Indonesia', 'Myanmar', 'T
hailand'])
```

```
In [763]: # list contains a list and tuple in it

          africa = sets.Set(['south Africa', 'Mozambique', ['Moracco', 'tunisia'], ('ken
          ya', 'sudan')])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-763-a1cb69f5b50e> in <module>()
      1 # list contains a list and tuple in it
      2
----> 3 africa = sets.Set(['south Africa', 'Mozambique', ['Moracco', 'tunisi
a'], ('kenya', 'sudan')])

c:\python27\lib\sets.pyc in __init__(self, iterable)
    412         self._data = {}
    413         if iterable is not None:
--> 414             self._update(iterable)
    415
    416     def __getstate__(self):

c:\python27\lib\sets.pyc in _update(self, iterable)
    357                 try:
    358                     for element in it:
--> 359                         data[element] = value
    360                     return
    361                 except TypeError:

TypeError: unhashable type: 'list'
```

```
In [764]: africa = sets.Set(['south Africa', 'Mozambique', ('Moracco', 'tunisia'), ('ken
          ya', 'sudan')])
```

```
In [765]: print africa
```

```
Set([('Moracco', 'tunisia'), ('kenya', 'sudan'), 'Mozambique', 'south Afric
a'])
```

```
In [766]: engineers = set(['John', 'Jane', 'Jack', 'Janice'])
          programmers = sets.Set({'Jack', 'Sam', 'Susan', 'Janice'})
          managers  = {'Jane', 'Jack', 'Susan', 'Zack'}

          print type(engineers), type(programmers), type(managers)
```

```
<type 'set'> <class 'sets.Set'> <type 'set'>
```

```
In [767]: programmers = set(programmers)

          print type(programmers)
```

```
<type 'set'>
```

4.3.2 Set Operations

```
|  - union operator
&  - Intersection operator
-  - difference operator
```

```
In [768]: employees = engineers | programmers | managers

          print employees
```

```
set(['Jack', 'Sam', 'Susan', 'Jane', 'Janice', 'John', 'Zack'])
```

```
In [769]: engg_managers = engineers & managers

          print engg_managers
```

```
set(['Jane', 'Jack'])
```

```
In [770]: onlyManagers = managers - engineers - programmers # same as (managers - engine
          ers) - programmers

          print onlyManagers
```

```
set(['Zack'])
```

```
In [771]: onlyEngineers = engineers - managers - programmers

          print onlyEngineers
```

```
set(['John'])
```

4.3.3 Mutability of sets

```
In [772]: engineers.add('Naseer')  # add - to add an element to the set
```

```
In [773]: print engineers
```

```
set(['Jane', 'Naseer', 'Janice', 'John', 'Jack'])
```

4.3.4 SuperSet and Subset

```
In [774]: employees.issuperset(engineers)
```

```
Out[774]: False
```

```
In [775]: engineers.issuperset(employees)
Out[775]: False
```

```
In [776]: print engineers, '\n', employees

          set(['Jane', 'Naseer', 'Janice', 'John', 'Jack'])
          set(['Jack', 'Sam', 'Susan', 'Jane', 'Janice', 'John', 'Zack'])
```

```
In [777]: employees.add('Naseer')
```

```
In [778]: employees.issuperset(engineers)

          # It means that every element of 'engineers' set is present in 'employees' set
Out[778]: True
```

```
In [779]: employees.discard('Susan')
```

```
In [780]: print employees

          set(['Naseer', 'Jack', 'Sam', 'Jane', 'Janice', 'John', 'Zack'])
```

Interview Question : what is the error that occurs for set.discard(element), when the element is not present in the set

```
In [781]: employees.discard('Shoban')
```

Observation: didn't result any execption, even though 'Shoban' is not present in set

4.3.4 frozenset

- set is a mutable object; elements in a set can be modified
- frozenset is an immutable object; elements in a frozenset can't be modified

```
In [782]: vetoCountries = set(['US', 'UK', 'Russia', 'China', 'France'])

          print vetoCountries

          set(['France', 'China', 'UK', 'US', 'Russia'])
```

```
In [783]: print dir(vetoCountries)
```

```
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__iand__', '__init__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le
__', '__len__', '__lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduc
e__', '__reduce_ex__', '__repr__', '__ror__', '__rsub__', '__rxor__', '__seta
ttr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__xor__', 'a
dd', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersec
tion', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop',
 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'u
pdate']
```

```
In [784]: vetoCountries = frozenset(['US', 'UK', 'Russia', 'China', 'France'])

          print vetoCountries
```

```
frozenset(['France', 'China', 'UK', 'US', 'Russia'])
```

```
In [785]: print dir(vetoCountries)
```

```
['__and__', '__class__', '__cmp__', '__contains__', '__delattr__', '__doc__',
 '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__',
 '__init__', '__iter__', '__le__', '__len__', '__lt__', '__ne__', '__new__',
 '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__',
 '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__',
 '__subclasshook__', '__xor__', 'copy', 'difference', 'intersection', 'isdisj
oint', 'issubset', 'issuperset', 'symmetric_difference', 'union']
```

```
In [786]: testSet  = {12, 12.34, True, (45.67, (23, 45)), 'Omen', 2 + 3j,
          set(['France','Russia'])}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-786-0dec7e97501d> in <module>()
----> 1 testSet  = {12, 12.34, True, (45.67, (23, 45)), 'Omen', 2+3j, set(['F
rance','Russia'])}

TypeError: unhashable type: 'set'
```

```
In [787]: testSet  = {12, 12.34, True, (45.67, (23, 45)), 'Omen', 2 + 3j, frozenset(['Fr
          ance','Russia'])}

          print testSet, type(testSet)
```

```
set([True, (2+3j), 'Omen', 12.34, 12, frozenset(['Russia', 'France']), (45.6
7, (23, 45))]) <type 'set'>
```

NOTE: Sets are mutable (can be changed); frozensets are immutable (can't be changed). Both sets and frozensets can store immutable objects only.

As set is mutable, it can't be placed in a set; whereas as frozenset is immutable, it can be placed within a set or frozenset.

```
In [788]: fruits = {'Mango', 'Apple', 'Papaya', 'apple'}

          vegetables = {'Beetroot', 'cabbage', 'Carrot', 'Carrot'}
```

```
In [789]: fruitsAndVegetables = fruits.union(vegetables)

          print fruitsAndVegetables
```

```
set(['Beetroot', 'Mango', 'Papaya', 'apple', 'Carrot', 'cabbage', 'Apple'])
```

```
In [790]: fruits.update('tomato')
```

```
In [791]: print fruits
```

```
set(['a', 'Papaya', 'apple', 'm', 'o', 'Mango', 't', 'Apple'])
```

```
In [792]: fruits.update(['tomato'])
          fruits.update(('banana'))   # string
          fruits.update({'Banana'})

          print fruits
```

```
set(['a', 'tomato', 'b', 'Papaya', 'apple', 'm', 'o', 'n', 'Mango', 't', 'Ban
ana', 'Apple'])
```

```
In [793]: fruits.update(('banana',))  # tuple element

          print fruits
```

```
set(['a', 'tomato', 'b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango',
 't', 'Banana', 'Apple'])
```

```
In [794]: fruits.discard('a')

          print fruits
```

```
set(['tomato', 'b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 't',
 'Banana', 'Apple'])
```

```
In [795]: fruits.discard('t')

          print fruits
```

```
set(['tomato', 'b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Ban
ana', 'Apple'])
```

```
In [796]: fruits.discard('m','o', 't')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-796-f92a0d176540> in <module>()
----> 1 fruits.discard('m','o', 't')

TypeError: discard() takes exactly one argument (3 given)
```

Inference: set.discard() will discards only one element once

```
In [797]: fruits.discard(('m','o', 't'))

          print fruits

          set(['tomato', 'b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Ban
          ana', 'Apple'])
```

Inference: set.discard() tries to locate and remove the tuple ('m','o', 't')

```
In [798]: fruits.intersection(vegetables)

Out[798]: set()

In [799]: vegetables.update(['tomato', 'watermelon'])

          print vegetables

          set(['Beetroot', 'tomato', 'cabbage', 'Carrot', 'watermelon'])

In [800]: fruits.intersection(vegetables)

Out[800]: {'tomato'}

In [801]: print fruits - vegetables

          set(['b', 'Papaya', 'apple', 'm', 'o', 'n', 'Mango', 'Banana', 'banana', 'App
          le'])
```

Observation: Observe that fruits.intersection(vegetables) elements are not present

```
In [802]: print vegetables - fruits

          set(['Beetroot', 'cabbage', 'Carrot', 'watermelon'])

In [803]: fruits.intersection(vegetables) == vegetables.intersection(fruits)

Out[803]: True

In [804]: fruits - vegetables  != vegetables - fruits

Out[804]: True
```

NOTE: set difference is not Commutative; whereas intersection attribute of set is Commutative. Intersection results in the common elements among the sets given

```
In [805]: fruits.isdisjoint(vegetables)  # no, there is a common element

Out[805]: False
```

```
In [806]: fruits.isdisjoint(vetoCountries)  #yes, there is no common element

Out[806]: True
```

```
In [807]: print fruits

          print fruits.pop()   # It will remove some element in random

          print fruits

          set(['tomato', 'b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Ban
          ana', 'Apple'])
          tomato
          set(['b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Banana', 'App
          le'])
```

```
In [808]: print fruits
          print fruits.remove('b')   # To remove a particular element; set.remove() will
           not return anything
          print fruits

          set(['b', 'Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Banana', 'App
          le'])
          None
          set(['Papaya', 'apple', 'banana', 'm', 'o', 'n', 'Mango', 'Banana', 'Apple'])
```

Assignment : Explore the differences between set.remove() vs set.discard() vs set.pop()

Assignment : Explore the differences between set.update() and set.add()

Interview Question : what is the difference between discard, pop and remove methods of set.

```
In [809]: print asean

          Set(['Malaysia', 'Philiphines', 'Indonesia', 'Myanmar', 'Thailand'])
```

```
In [810]: print asean.pop()  # deletes an elemnt, in random; So, not preferred

          Malaysia
```

```
In [811]: asean.remove('Myanmar') # delete the given element
```

NOTE: remove() can't delete multiple elements

In [812]:
```
print asean
```
```
Set(['Philiphines', 'Indonesia', 'Thailand'])
```

In [813]:
```
asean.remove('Myanmar') # delete the given element
```
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-813-fe684ca56941> in <module>()
----> 1 asean.remove('Myanmar') # delete the given element

c:\python27\lib\sets.pyc in remove(self, element)
    516         """
    517         try:
--> 518             del self._data[element]
    519         except TypeError:
    520             transform = getattr(element, "__as_temporarily_immutable_
_", None)

KeyError: 'Myanmar'
```

In [814]:
```
asean.discard('Myanmar')
```

Inference: set.discard() will not throw any exception, even if the quieried element is not present

Assignment : try the set.remove(), set.discard() and set.pop() with multiple elements, and with string in different collections

Conclusion: set.remove() vs set.discard() vs set.pop()

```
    set.remove()   - Used to remove an element. Throws KeyError, if the specifed el
ement is not present
    set.discard()  - Used to remove an element. Doesn't raise any error, if specifi
ed element is not present.
    set.pop()      - Used to remove and RETURN a random element from the set. Raise
s keyError, if no element is present.
```

for sets A and B,

```
symmetric difference is (A-B) | (B-A)
```

It is same as union - intersection

```
In [815]: fruits = {'Mango', 'Apple', 'Papaya', 'tomato'}

          vegetables = {'Beetroot', 'cabbage', 'tomato', 'Carrot', 'Carrot'}
```

```
In [816]: fruits.symmetric_difference(vegetables)

Out[816]: {'Apple', 'Beetroot', 'Carrot', 'Mango', 'Papaya', 'cabbage'}
```

```
In [817]: # set.symmetric_difference() is cumulative

          fruits.symmetric_difference(vegetables) == vegetables.symmetric_difference(fru
          its)

Out[817]: True
```

Interview Question : what is the result of set1 = {1, 'Python', True}

```
In [818]: set1 = {1, 'Python', True}

          print type(set1), set1

          <type 'set'> set(['Python', True])
```

```
In [819]: id(1), id(True)

Out[819]: (2372312, 505422916)
```

NOTE: True is preferred as it is built-in object

```
In [820]: print all(set1)

          True
```

```
In [821]: print any(set1)

          True
```

```
In [822]: set2 = {10, 10.9, 0.01, 0, 'Prog', None, ''}
```

```
In [823]: print all(set2)

          False
```

```
In [824]: print any(set2)

          True
```

```
In [825]: for i in set2:
              print i,

           0 10 None 10.9 Prog 0.01
```

```
In [826]: lc = [i for i in set2]    # list comprehension
          print type(lc), lc

          <type 'list'> ['', 0, 10, None, 10.9, 'Prog', 0.01]
```

```
In [827]: sc = {i for i in set2}    # set comprehension
          print type(sc), sc

          <type 'set'> set(['', 0, 10, None, 10.9, 'Prog', 0.01])
```

enumerate() - builtin function; enumerate() stores the index of the elements iterated.

```
In [828]: for i,j in enumerate(set2):
              print i,j

          0
          1 0
          2 10
          3 None
          4 10.9
          5 Prog
          6 0.01
```

```
In [829]: max(set2), min(set2)

Out[829]: ('Prog', None)
```

```
In [830]: a = None; print type(a)

          <type 'NoneType'>
```

```
In [831]: sorted(set2)

Out[831]: [None, 0, 0.01, 10, 10.9, '', 'Prog']
```

```
In [832]: len(set2)

Out[832]: 7
```

4.3.6 Orderedset

- Used to store elements in an ascending order
- This is a module, to be imported
- This module doesn;t come with standard library
- It must be installed using the command

> pip install orderedset

NOTE: If you are working in windows, and the error is as below,

```
error: Microsoft Visual C++ 9.0 is required (Unable to find vcvarsall.bat). Get it
 from http://aka.ms/vcpython27
```

then, download VC++ compiler from http://aka.ms/vcpython27 (http://aka.ms/vcpython27) and install.

In [833]:
```
import orderedset
```

In [834]:
```
oset = orderedset.OrderedSet([11, 2, 3.33, '1', 1, 'python', frozenset('tomat
o')])

print type(oset)
print oset     # orderedset.OrderedSet() ensures that the assigned order of th
e set is retained.
```

```
<class 'orderedset._orderedset.OrderedSet'>
OrderedSet([11, 2, 3.33, '1', 1, 'python', frozenset(['a', 'm', 't', 'o'])])
```

In [835]:
```
print dir(oset)
```

```
['__abstractmethods__', '__and__', '__class__', '__contains__', '__delattr_
_', '__dict__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute_
_', '__getitem__', '__gt__', '__hash__', '__iand__', '__init__', '__ior__',
 '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '__lt__', '__metacl
ass__', '__module__', '__ne__', '__new__', '__or__', '__pyx_vtable__', '__qua
lname__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed_
_', '__ror__', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str_
_', '__sub__', '__subclasshook__', '__weakref__', '__xor__', '_abc_cache', '_
abc_negative_cache', '_abc_negative_cache_version', '_abc_registry', '_from_i
terable', '_hash', 'add', 'clear', 'copy', 'difference', 'difference_update',
 'discard', 'index', 'intersection', 'intersection_update', 'isdisjoint', 'is
orderedsubset', 'isorderedsuperset', 'issubset', 'issuperset', 'pop', 'remov
e', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']
```

In [836]:
```
oset.update('AI')
```

In [837]:
```
print oset    # Observe that updated elements are placed in the end (right-mos
t)
```

```
OrderedSet([11, 2, 3.33, '1', 1, 'python', frozenset(['a', 'm', 't', 'o']),
 'A', 'I'])
```

In [838]:
```
oset.update(['AI'])

print oset
```

```
OrderedSet([11, 2, 3.33, '1', 1, 'python', frozenset(['a', 'm', 't', 'o']),
 'A', 'I', 'AI'])
```

In [839]:
```
print oset.pop()
```

```
AI
```

```
In [840]: print oset.pop()
```

```
I
```

```
In [841]: for _ in range(4): print oset.pop()
```

```
A
frozenset(['a', 'm', 't', 'o'])
python
1
```

```
In [842]: print oset  # Observe that four set elements were removed from the last(right-
          most)
```

```
OrderedSet([11, 2, 3.33, '1'])
```

Assignment : Try to get a sorted set from the given set, using orderedset module

4.4 Dictionaries

- It is a key/value structure
- It contain series of key:value pair, separated by comma(,) operator and enclosed in {} .

> eg: dict1 = {key1:value1, key2: value2}

- It doesn't store duplicate keys
- The keys in dictionaries should be immutable (string, tuple, int, float, frozenset). whereas list, set is not possible.
- Indexing is done based on keys, and not position.

```
In [843]: d = {}             # Empty dictionary

          print type(d), d
```

```
<type 'dict'> {}
```

```
In [844]: d1 = {12}

          print type(d1)
```

```
<type 'set'>
```

Interview Question: what is the type of 'd2' in the below statement:

```
d2 = {12,}
```

```
In [845]: d2 = {12,}

          print type(d2)

          <type 'set'>
```

```
In [846]: d3 = {12:23}

          print type(d3)

          <type 'dict'>
```

```
In [847]: d4 = {'a': 'apple', 'b': 'banana', 'c': 'cat'}   # method 1 of dictionary crea
          tion

          print d4                                         # Observe the order of the el
          ements

          {'a': 'apple', 'c': 'cat', 'b': 'banana'}
```

```
In [848]: dict1 = {}                                      # method 2 of dictionaru creat
          ion
          dict1['A'] = 'Apple'
          dict1['B'] = 'Banana'                           # The dictionary keys must be
           unique
          dict1['B'] = 'Ball' # For duplicate assignments for a key, the key will retain
           the last assignment
          dict1['c'] = 'Cat'

          print dict1

          {'A': 'Apple', 'c': 'Cat', 'B': 'Ball'}
```

Indexing the dictionaries

```
In [849]: print "There is  one ", dict1['c'], " in the hall"

          There is  one  Cat  in the hall
```

```
In [850]: print dict1['C']      # case sensitivity

          ---------------------------------------------------------------------------
          KeyError                                  Traceback (most recent call last)
          <ipython-input-850-c11606669e7d> in <module>()
          ----> 1 print dict1['C']      # case sensitivity

          KeyError: 'C'
```

```
In [851]: 'C' in dict1  # membership check

Out[851]: False
```

```
In [852]: 'c' in dict1  # membership check

Out[852]: True

In [853]: dict1.values()

Out[853]: ['Apple', 'Cat', 'Ball']

In [854]: dict1.keys()

Out[854]: ['A', 'c', 'B']
```

Interview Question : what is the type of the result of dictionary.items()?

```
In [855]: dict1.items()

Out[855]: [('A', 'Apple'), ('c', 'Cat'), ('B', 'Ball')]

In [856]: dict1

Out[856]: {'A': 'Apple', 'B': 'Ball', 'c': 'Cat'}
```

Editing an existing dictionary

```
In [857]: dict1['ac'] = 'Air Conditioner'

In [858]: dict1['z'] = 'Zombie'

In [859]: print dict1

          {'A': 'Apple', 'c': 'Cat', 'B': 'Ball', 'ac': 'Air Conditioner', 'z': 'Zombi
          e'}

In [860]: if 'ac' in dict1:
              print 'There is ', dict1['ac']

          There is  Air Conditioner

In [861]: dict1.get('B')

Out[861]: 'Ball'

In [862]: dict1.get('b')  # Doesn't raise any error, in the absence of specified key

In [863]: dict1.get('b', 'xxx')  # if not present, returns the specified value

Out[863]: 'xxx'
```

iterations on dictionaries

```
In [864]: [i for i in dict1.items()]

Out[864]: [('A', 'Apple'),
           ('c', 'Cat'),
           ('B', 'Ball'),
           ('ac', 'Air Conditioner'),
           ('z', 'Zombie')]

In [865]: [i for i in dict1.keys()]

Out[865]: ['A', 'c', 'B', 'ac', 'z']

In [866]: [i for i in dict1.values()]

Out[866]: ['Apple', 'Cat', 'Ball', 'Air Conditioner', 'Zombie']

In [867]: [i for i in dict1]  # In loops, dict1 means dict1.keys() by default

Out[867]: ['A', 'c', 'B', 'ac', 'z']
```

NOTE: By default, iterating over a dictionary takes place on keys() only

String Formatting with dictionaries

```
In [868]: cricket ={'players': 'Batsmen and Bowlers', 'count': 11}

          print cricket

          {'count': 11, 'players': 'Batsmen and Bowlers'}

In [869]: cricket['players']

Out[869]: 'Batsmen and Bowlers'

In [870]: sentence = "The %(players)s in cricket team are %(count)d in number!"%cricket

          print sentence

          The Batsmen and Bowlers in cricket team are 11 in number!

In [871]: sentence = "The %(players)r in cricket team are %(count)r in number!"%cricket

          print sentence

          The 'Batsmen and Bowlers' in cricket team are 11 in number!
```

```
In [872]: sentence = "The %(0)r in cricket team are %(1)r in number!"%cricket.items()

          print sentence
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-872-766646641dec> in <module>()
----> 1 sentence = "The %(0)r in cricket team are %(1)r in number!"%cricket.i
tems()
      2 
      3 print sentence

TypeError: list indices must be integers, not str
```

NOTE: Dictionaries can't be indexed using positions

```
In [873]: cricket.items()
```

```
Out[873]: [('count', 11), ('players', 'Batsmen and Bowlers')]
```

```
In [874]: sentence = "The %s in cricket team are %d in number!"%cricket.items()[0]

          print sentence
```

```
The count in cricket team are 11 in number!
```

```
In [875]: sentence = "The %r in cricket team are %r in number!"%cricket.items()[0]

          print sentence
```

```
The 'count' in cricket team are 11 in number!
```

```
In [876]: for i in range(len(cricket.items())):
              tmp = cricket.items()[i]
              print "The %r in cricket team are %r in number!"%tmp
```

```
The 'count' in cricket team are 11 in number!
The 'players' in cricket team are 'Batsmen and Bowlers' in number!
```

But, there is logical error

```
In [877]: (cricket['players'], cricket['count'])
```

```
Out[877]: ('Batsmen and Bowlers', 11)
```

```
In [878]: print "The %r in cricket team are %r in number!"%(cricket['players'],
          cricket['count'])
```

```
The 'Batsmen and Bowlers' in cricket team are 11 in number!
```

```
In [879]: dict1.clear()   # results in empty dictionary object
In [880]: dict1

Out[880]: {}

In [881]: del dict1   # deletes the object

In [882]: dict1

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-882-8239e7494a4a> in <module>()
          ----> 1 dict1

          NameError: name 'dict1' is not defined
```

dictonary keys should be immutable

Interview Question : what is the result of the below statement:

```
dict1 = {[1,2,3]: 'numbers'}
```

```
In [883]: dict1 = {[1,2,3]: 'numbers'}

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-883-b12b39d823ae> in <module>()
          ----> 1 dict1 = {[1,2,3]: 'numbers'}

          TypeError: unhashable type: 'list'

In [884]: dict1 = {(1,2,3): 'numbers'}  #possible, as tuple is immutable

In [885]: dict1 = {"1,2,3": 'numbers'}  # possible, as string is immutable

In [886]: dict1 = {123: 'numbers'}  # possible, as int is immutable

In [887]: dict1

Out[887]: {123: 'numbers'}

In [888]: dict1 = {{1,2,3}: 'numbers'}  # Not possible, as set is mutable

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-888-7755ecdb2543> in <module>()
          ----> 1 dict1 = {{1,2,3}: 'numbers'}  # Not possible, as set is mutable

          TypeError: unhashable type: 'set'
```

```
In [889]: dict1 = {frozenset({1,2,3}): 'numbers'}  #  possible, as frozen set is immutab
          le
```

```
In [890]: dict1
```

```
Out[890]: {frozenset({1, 2, 3}): 'numbers'}
```

```
In [891]: dict1 = {3: 'numbers'}  # on integers
```

```
In [892]: dict1 = {3.333: 'numbers'}  # on floats
```

```
In [893]: dict1 = {True: 'numbers'}  # on booleans
```

Interview Question : what is the result of dict1 = {True: 'numbers', 1: 'one', 2.0: 'two', 3.333: 'three'}

```
In [894]: dict1 = {True: 'numbers', 1: 'one', 2.0: 'two', 2.0: '2.0', 3.333: 'three'}
```

```
In [895]: dict1
```

```
Out[895]: {True: 'one', 2.0: '2.0', 3.333: 'three'}
```

NOTE: As 'True' is a python object, it is preferred.

COPY in dictionaries

```
In [896]: dictHardCopy = dict1 # Hard COPY (assignment operation)
```

```
In [897]: dictCopy = dict1.copy()  # soft COPY
```

```
In [898]: print dict1, '\n', dictHardCopy, '\n', dictCopy
```

```
{True: 'one', 2.0: '2.0', 3.333: 'three'}
{True: 'one', 2.0: '2.0', 3.333: 'three'}
{True: 'one', 2.0: '2.0', 3.333: 'three'}
```

```
In [899]: dict1 == dictHardCopy == dictCopy
```

```
Out[899]: True
```

```
In [900]: dict1 is dictHardCopy is dictCopy
```

```
Out[900]: False
```

```
In [901]: dictHardCopy[3.333] = '3333333'  # updating the key, not position
```

```
In [902]: dictCopy[2.0] = '2222222'  # indexing is done with key, and not index here
In [903]: print dict1, '\n', dictHardCopy, '\n', dictCopy

          {True: 'one', 2.0: '2.0', 3.333: '3333333'}
          {True: 'one', 2.0: '2.0', 3.333: '3333333'}
          {True: 'one', 2.0: '2222222', 3.333: 'three'}

In [904]: mDict = {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333:
          'three'} } }

In [905]: mdictCopy =  mDict.copy()  # soft COPY

In [906]: mDict[3.333][3.333][3.333]

Out[906]: 'three'

In [907]: mDict[3.333][3.333][3.333] = 9999

In [908]: print mDict, '\n', mdictCopy

          {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333: 9999}}}
          {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333: 9999}}}

In [909]: import copy

          mDictDeepCopy = copy.deepcopy(mDict)

In [910]: print mDict, '\n', mDictDeepCopy

          {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333: 9999}}}
          {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333: 9999}}}

In [911]: mDict[3.333][3.333][True]

Out[911]: 'one'

In [912]: mDict[3.333][3.333][1]      #Observe that both are resulting the same.

Out[912]: 'one'

In [913]: mDict[3.333][3.333][True] = 1111

In [914]: print mDict, '\n', mDictDeepCopy

          {True: 'one', 3.333: {True: 'one', 3.333: {True: 1111, 3.333: 9999}}}
          {True: 'one', 3.333: {True: 'one', 3.333: {True: 'one', 3.333: 9999}}}

In [915]: sorted(mDict)

Out[915]: [True, 3.333]
```

```
In [916]: sorted(mDict.keys())

Out[916]: [True, 3.333]
```

```
In [917]: sorted(mDict.values())

Out[917]: [{True: 'one', 3.333: {True: 1111, 3.333: 9999}}, 'one']
```

```
In [918]: sorted(mDict.items())

Out[918]: [(True, 'one'), (3.333, {True: 'one', 3.333: {True: 1111, 3.333: 9999}})]
```

```
In [919]: fruits = {'a': 'apple', 'b': 'banana', 'd': 'donut'} # observe the common keys
           in fruits and countries

          countries = {'a': 'america', 'b': 'bahamas', 'c':'canada'}
```

```
In [920]: print fruits

          {'a': 'apple', 'b': 'banana', 'd': 'donut'}
```

```
In [921]: fruits.update(countries)
```

```
In [922]: print fruits

          {'a': 'america', 'c': 'canada', 'b': 'bahamas', 'd': 'donut'}
```

```
In [923]: fruits = {'a': 'apple', 'b': 'banana', 'd': 'donut'}   # re-initializing
```

```
In [924]: print countries

          {'a': 'america', 'c': 'canada', 'b': 'bahamas'}
```

```
In [925]: countries.update(fruits)
          print countries             # observe that as there is no key 'c' in fruits, i
          t is not updated

          {'a': 'apple', 'c': 'canada', 'b': 'banana', 'd': 'donut'}
```

creating dictionaries from lists

```
In [926]: countries = ['India', 'US', 'UK', 'Germany']

          capitals = ['New Delhi', 'Washington', 'London', 'Berlin']
```

```
In [927]: countriesNcaptitals = zip(countries, capitals)   # zip() builtin function; ret
          urns list of tuples
          print countriesNcaptitals
          print type(countriesNcaptitals)

          [('India', 'New Delhi'), ('US', 'Washington'), ('UK', 'London'), ('Germany',
           'Berlin')]
          <type 'list'>
```

Interview Question : How to create a list of tuples?

Ans: Using zip, map

```
In [928]: cncDictionary = dict(countriesNcaptitals)  # dict() builtin function to create
           dictionary
          print type(cncDictionary)
          print cncDictionary

          <type 'dict'>
          {'Germany': 'Berlin', 'India': 'New Delhi', 'UK': 'London', 'US': 'Washingto
          n'}
```

```
In [929]: dict(zip(['India', 'US', 'UK', 'Germany'], ['New Delhi', 'Washington', 'Londo
          n', 'Berlin']))

Out[929]: {'Germany': 'Berlin', 'India': 'New Delhi', 'UK': 'London', 'US': 'Washingto
          n'}
```

```
In [930]: d1 = {'a': 1, 'b': 2, 'c': 3}
          d2 = {'a': 1, 'b': 2, 'c': 3}

          cmp(d1,d2)

Out[930]: 0
```

```
In [931]: d2['d'] = 123;
          print d2

          {'a': 1, 'c': 3, 'b': 2, 'd': 123}
```

```
In [932]: cmp(d1, d2)

Out[932]: -1
```

```
In [933]: d1['d'] = '123'
          d1['e'] = '345'

          cmp(d1,d2)

Out[933]: 1
```

```
In [934]: len(d1), len(d2)

Out[934]: (5, 4)

In [935]: print d1

          {'a': 1, 'c': 3, 'b': 2, 'e': '345', 'd': '123'}

In [936]: d4 ={}
          d4.fromkeys(d1)  # to extract the keys of d1, and place them for d4

Out[936]: {'a': None, 'b': None, 'c': None, 'd': None, 'e': None}

In [937]: d5 ={}
          d5.fromkeys(d1, 'Python') # To place a default value, instead of None

Out[937]: {'a': 'Python', 'b': 'Python', 'c': 'Python', 'd': 'Python', 'e': 'Python'}
```

NOTE: dictionary Values can't be extracted in the same way

```
In [938]: print dir(d1)

          ['__class__', '__cmp__', '__contains__', '__delattr__', '__delitem__', '__doc
          __', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__
          gt__', '__hash__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__
          ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',
           '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'clear', 'copy',
           'fromkeys', 'get', 'has_key', 'items', 'iteritems', 'iterkeys', 'itervalue
          s', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values', 'viewitems',
           'viewkeys', 'viewvalues']

In [939]: print d1

          {'a': 1, 'c': 3, 'b': 2, 'e': '345', 'd': '123'}

In [940]: d1.get('a')

Out[940]: 1

In [941]: d1.get('A')

In [942]: d1.get('A', 'Not present')

Out[942]: 'Not present'

In [943]: d1.has_key('a')

Out[943]: True

In [944]: d1.has_key('A')

Out[944]: False
```

Interview Question: what is the difference between dictionary attributes: pop() and popitem()

Assignment : Write a script to take the names of our five friends in a list, and their designations in a separate list. ... Then, create a dictionary, containing their name, designation pairs

Assignment : Write a script to get the letter frequency from a given sentence. Display the top three most occurred letters

```
In [945]: d1 = {'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '345'}

In [946]: d1.setdefault('a', None)  # works same as indexing a key, when key is present

Out[946]: 1

In [947]: d1

Out[947]: {'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '345'}

In [948]: d1.setdefault('A', None)  # works same as indexing a key, when key is present

In [949]: d1

Out[949]: {'A': None, 'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '345'}

In [950]: d1.setdefault('A', 'not present')  # As 'A' is present, None will be returned.

In [951]: d1

Out[951]: {'A': None, 'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '345'}

In [952]: d1.setdefault('B', 'not present')

Out[952]: 'not present'

In [953]: d1

Out[953]: {'A': None, 'B': 'not present', 'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '34
          5'}

In [954]: d1.setdefault('B', 'someThing')

Out[954]: 'not present'

In [955]: d1

Out[955]: {'A': None, 'B': 'not present', 'a': 1, 'b': 2, 'c': 3, 'd': '123', 'e': '34
          5'}
```

```
In [956]: d1.values()     # results a list
Out[956]: [1, None, 3, 2, '345', '123', 'not present']
```

```
In [957]: print type(d1.values())
          <type 'list'>
```

```
In [958]: d1.viewvalues()   # results a dict_item
Out[958]: dict_values([1, None, 3, 2, '345', '123', 'not present'])
```

```
In [959]: print type(d1.viewvalues())
          <type 'dict_values'>
```

```
In [960]: d1.items()
Out[960]: [('a', 1),
           ('A', None),
           ('c', 3),
           ('b', 2),
           ('e', '345'),
           ('d', '123'),
           ('B', 'not present')]
```

```
In [961]: d1.viewitems()
Out[961]: dict_items([('a', 1), ('A', None), ('c', 3), ('b', 2), ('e', '345'), ('d', '1
          23'), ('B', 'not present')])
```

```
In [962]: dictContent = d1.iteritems()  # returns items as a generator; need to iterate
           with next() to get the items

          print type(dictContent)
          <type 'dictionary-itemiterator'>
```

```
In [963]: print dictContent
          <dictionary-itemiterator object at 0x03E45D20>
```

```
In [964]: print dictContent.next()
          ('a', 1)
```

d1.iterkeys() and d1.itervalues() will work in the same way.

```
In [965]: d1= {True: 'one', 2: '2222', 3: 'three', 'b': 'ball'}
```

```
In [966]: print d1.pop(2)   # 2 here is key, not position
          2222
```

```
In [967]: print d1.pop()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-967-16d02b4d0b6a> in <module>()
----> 1 print d1.pop()

TypeError: pop expected at least 1 arguments, got 0
```

```
In [968]: print d1.pop('abcd')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-968-6973f3cda490> in <module>()
----> 1 print d1.pop('abcd')

KeyError: 'abcd'
```

```
In [969]: print d1.pop('abcd', None) # returns None, if the key is not present
```

```
None
```

```
In [970]: print d1.pop('abcd', "No Such Key") # To return default statement, in the abse
          nce of key
```

```
No Such Key
```

```
In [971]: d1 = {True: 'one', 2: '2222', 3: 'three', 'b': 'ball'}
```

```
In [972]: key,value = d1.popitem()   # deletes a random key-pair and retuns them

          print key, value
```

```
True one
```

```
In [973]: d1.popitem()
```

```
Out[973]: (2, '2222')
```

```
In [974]: key,value = d1.popitem()

          print key,value
```

```
3 three
```

```
In [975]: key,value = d1.popitem(); print key,value
```

```
b ball
```

```
In [976]: key,value = d1.popitem(); print key,value

          ---------------------------------------------------------------------------
          KeyError                                  Traceback (most recent call last)
          <ipython-input-976-b91033ef2679> in <module>()
          ----> 1 key,value = d1.popitem(); print key,value

          KeyError: 'popitem(): dictionary is empty'

In [977]: d = {'a':1, 'b':2}

In [978]: d.get('a', 34)   # As key 'a' is present, it is resulting in corresponding val
          ue

Out[978]: 1

In [979]: d.get('z', 34)   # In the absence of key 'z', it results the second argument

Out[979]: 34

In [980]: print d         # Observe that 'z' is not created

          {'a': 1, 'b': 2}

In [981]: d.setdefault('z', 34)

Out[981]: 34

In [982]: print d         # Observe that 'z' is  created

          {'a': 1, 'b': 2, 'z': 34}
```

In [983]:

```python
#!/usr/bin/python
"""
        Purpose: To count the number of times, each character occurred in the
 sentence.
        Output: Each character and its occurrence count, as a pair.
"""
# characterFrequencyAnalysis.py

#sentence = "It always seem impossible, until it is achieved!"
sentence = raw_input("Enter a Quote: ")
count = {} # empty dictionary

for character in sentence:
     count[character] = count.get(character, 0) + 1

print "character: occurrenceFrequency \n"

#print count

#for key,value in count.items():
#    print key, value

for item in count.items():
        print item

#for index, item in enumerate(count.items()):
#    print index, item
```

```
Enter a Quote: It always seem impossible, until it is achieved!
character: occurrenceFrequency 

('!', 1)
(' ', 7)
(',', 1)
('I', 1)
('a', 3)
('c', 1)
('b', 1)
('e', 5)
('d', 1)
('i', 6)
('h', 1)
('m', 2)
('l', 3)
('o', 1)
('n', 1)
('p', 1)
('s', 5)
('u', 1)
('t', 3)
('w', 1)
('v', 1)
('y', 1)
```

Assignment : In this characterFrequencyAnalysis.py example, try to display first three most frequently occurred characters

Memoization

- To store the values which are already compiled, in cache, to optimize the time consumption

Interview Question : What is memoization. How to achieve it in dictionaries?

In [984]:
```
alreadyknown = {0: 0, 1: 1, 2: 1, 3: 2, 4: 3, 5: 5}      # Memoization

def fib(n):
    if n not in alreadyknown:
        new_value = fib(n-1) + fib(n-2)
        alreadyknown[n] = new_value
    return alreadyknown[n]

print "fib(20) = ", fib(20)

# fib(20)
#alreadyknown[20] = fib(19) + fib(18)
```

```
fib(20) =  6765
```

Ordered Dictionary

In [985]:
```
d = {'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}  # regular unsorted dict
ionary
print d
```

```
{'orange': 2, 'pear': 1, 'banana': 3, 'apple': 4}
```

In [986]:
```
import collections
```

In [987]:
```
collections.OrderedDict({'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2})
```

Out[987]:
```
OrderedDict([('orange', 2), ('pear', 1), ('banana', 3), ('apple', 4)])
```

In [988]:
```
collections.OrderedDict(sorted(d.items(), key=lambda t: t[0])) # Sorted by key
          [(k1,v1),(k2,v2)]
```

Out[988]:
```
OrderedDict([('apple', 4), ('banana', 3), ('orange', 2), ('pear', 1)])
```

In [989]:
```
collections.OrderedDict(sorted(d.items(), key=lambda t: t[1])) # Sorted by Val
ue
```

Out[989]:
```
OrderedDict([('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)])
```

Interview Question : How to sort a dictionary based on the length of the key?

```
In [990]: collections.OrderedDict(sorted(d.items(), key=lambda t: len(t[0]))) # Sorted b
          y length of key .

Out[990]: OrderedDict([('pear', 1), ('apple', 4), ('orange', 2), ('banana', 3)])
```

```
In [991]: collections.OrderedDict(sorted(d.items(), key=lambda t: len(str(t[1])))) # Sor
          ted by length of Value; Observe no change

Out[991]: OrderedDict([('orange', 2), ('pear', 1), ('banana', 3), ('apple', 4)])
```

```
In [992]: collections.OrderedDict(d.items())

Out[992]: OrderedDict([('orange', 2), ('pear', 1), ('banana', 3), ('apple', 4)])
```

Assignment : Use collections.OrdererDict to get an ordered dictionary. Try to do some example

5.0 Functions

- To reduce the code duplication for repetive logic
- Modularizing the complex problem, into simpler pieces.
- For better code reusage
- Information Hiding
- Functions in Python are called **First-class citizens**.
  - Function object have equal status as other objects
  - Functions too, can be assigned to variables, stored in collections (list, tuple, dictionary, set) or can be passed as arguments.
- Functions are of two types:
  1. Built-functions
  2. user-defined functions

Syntax:

```
def functionName(argument1, arguments2, ... ):
    statement1
    statement2
    ....
    return expression/Value  # return is optional, None object goes when return is
 not present.
```

Functions without input arguments

```
In [993]: def helloWorld():                 # function definition
              print "Hello World!!!"
```

```
In [994]: type(helloWorld)
```

```
Out[994]: function
```

```
In [995]: helloWorld()                      # function call
```

```
Hello World!!!
```

```
In [996]: hw = helloWorld()                # assigning the result of function call
          print type(hw), hw
```

```
Hello World!!!
<type 'NoneType'> None
```

NOTE: Observe that the result(assignment) is NoneType beacuse the default return is NoneType.

Problem : write a function to get all the team members for a new project in run-time, and display them along with their count.

```
In [997]: def teamMembers():
              team = raw_input('Enter the names of team members: ')
              print team, type(team)

          teamMembers()                  # function call
```

```
Enter the names of team members: Shoban, Shankar, Naseer Shiek
Shoban, Shankar, Naseer Shiek <type 'str'>
```

```
In [998]: def teamMembers():
              team = raw_input('Enter the names of team members: ')
              print team, type(team)
              print len(team)
              print team.split(',')
              print "There are ", len(team.split(',')), " members in the team"

          teamMembers()                  # function call
```

```
Enter the names of team members: Shoban, Shankar, Naseer Shiek
Shoban, Shankar, Naseer Shiek <type 'str'>
29
['Shoban', ' Shankar', ' Naseer Shiek']
There are  3  members in the team
```

Functions with inputs

Problem : Write a script to print the greater number, among two numbers

```
In [999]: def greaterNumber(a,b):
              if a > b:
                  print "%d is greater than %d"%(a,b)
              elif a < b:
                  print "%d is lesser than %d"%(a,b)
              else:
                  print "%d is equal to %d"%(a,b)

          greaterNumber(-32, -12)
          greaterNumber(32, -12)
          greaterNumber(-32, 12)
```

```
-32 is lesser than -12
32 is greater than -12
-32 is lesser than 12
```

Problem : Displaying a given name

```
In [1000]: def displayName(name):
               print "The name of the candidate is %r"%(name)

           displayName('Naseer')
```

```
The name of the candidate is 'Naseer'
```

Scope: Local and Global Variables

```
In [1001]: a = 10           # 'a' is an integer - immutable

           print "Initially, before going to function, a = %d"%(a)

           def localEx(a):
               print "In localEx() function, a = %d"%(a)
               a= 5
               print "In localEx() function, a = %d"%(a)

           localEx(a)
           print "Outside: a = %d"%(a)
```

```
Initially, before going to function, a = 10
In localEx() function, a = 10
In localEx() function, a = 5
Outside: a = 10
```

Inference: Changes made to the variable within function, does affect outside it

Interview Question : Explain the difference between global and local values, with an example?

global : It is a keyword to reflect the local changes (within the function) to the global level

In [1002]:
```
global a             # global declaration
a = 10
def globalEx():             # Observe that 'a' is not passed as input
    global a         # global declaration
    print "In globalEx() function, a = %d"%(a)
    a= 5
    print "In globalEx() function, a = %d"%(a)

globalEx()
print "Outside: a = %d"%(a)
```

```
In globalEx() function, a = 10
In globalEx() function, a = 5
Outside: a = 5
```

In [1003]:
```
myDict = {'a': 'Apple', 'b': 'Bat'}        # 'myDict' is dictionary - mutable

print "Initially, before going to function, myDict =", myDict

def localEx(a):
    print "In localEx() function, myDict =", myDict
    myDict['z'] = 'ZOO'
    print "In localEx() function, myDict =", myDict

localEx(a)
print "Outside:  myDict =", myDict
```

```
Initially, before going to function, myDict = {'a': 'Apple', 'b': 'Bat'}
In localEx() function, myDict = {'a': 'Apple', 'b': 'Bat'}
In localEx() function, myDict = {'a': 'Apple', 'b': 'Bat', 'z': 'ZOO'}
Outside:  myDict = {'a': 'Apple', 'b': 'Bat', 'z': 'ZOO'}
```

Inference: Changes made to the object within function, gets effected outside the function, if the object is of mutable type; else in immutable objects no change is reflected

Default and Keyword arguments

In [1004]:
```
def hello(name = "World!!!"):      # name is having a default input argument
    print "Hello ", name
```

In [1005]:
```
hello()                 # In the absence of input argument, It will display th
e default input argument
```

```
Hello  World!!!
```

```
In [1006]: hello('Shoban')

           Hello  Shoban
```

```
In [1007]: hello(name = 'Shoban Babu!!!')

           Hello  Shoban Babu!!!
```

```
In [1008]: def cricket(balls = 3, bats = 2):
                print "There are %d balls and %d bats"%(balls, bats)
```

```
In [1009]: cricket.func_defaults

Out[1009]: (3, 2)
```

```
In [1010]: cricket()

           There are 3 balls and 2 bats
```

```
In [1011]: cricket(4,5)    # taken based on position

           There are 4 balls and 5 bats
```

```
In [1012]: cricket(balls = 5, bats= 4)   # taken based on assignment

           There are 5 balls and 4 bats
```

```
In [1013]: cricket(bats = 5, balls= 40)   # taken based on assignment

           There are 40 balls and 5 bats
```

```
In [1014]: def nTimesDisplay(message, noOfTimes = 3):
                print message*noOfTimes
```

Here, as only one argument is default, the other should be given mandatorily.

NOTE: The default arguments must be placed in the last in the function definition.

```
In [1015]: nTimesDisplay('Python ')

           Python Python Python
```

```
In [1016]: nTimesDisplay('Python ', 5)

           Python Python Python Python Python
```

```
In [1017]: nTimesDisplay(5,'Python ')  # It identified based on the data type

           Python Python Python Python Python
```

```
In [1018]: nTimesDisplay(5.0,'Python ')

           ---------------------------------------------------------------------------
           TypeError                                 Traceback (most recent call last)
           <ipython-input-1018-9cea01eb6299> in <module>()
           ----> 1 nTimesDisplay(5.0,'Python ')

           <ipython-input-1014-dcd91946b6b5> in nTimesDisplay(message, noOfTimes)
                 1 def nTimesDisplay(message, noOfTimes = 3):
           ----> 2     print message*noOfTimes

           TypeError: can't multiply sequence by non-int of type 'float'
```

```
In [1019]: def funcExpressions(a,b = 12, c = 123):
               print "The value of a is %d"%(a)
               print "The value of b is %d"%(b)
               print "The value of c is %d"%(c)
```

```
In [1020]: funcExpressions(23)

           The value of a is 23
           The value of b is 12
           The value of c is 123
```

```
In [1021]: funcExpressions(23, c= 999)

           The value of a is 23
           The value of b is 12
           The value of c is 999
```

```
In [1022]: funcExpressions(c = 123, a = 234)

           The value of a is 234
           The value of b is 12
           The value of c is 123
```

return statement

- By default, user-defined functions will return 'None'

```
In [1023]: def evenOddTest(a):
               result = a % 2
               if result == 0:
                   return '%d is an even number'%(a)
               else:
                   return '%d is an odd number'%(a)
```

```
In [1024]: evenOddTest(223)

Out[1024]: '223 is an odd number'
```

```
In [1025]: print evenOddTest(223)
```
```
223 is an odd number
```
```
In [1026]: eot = evenOddTest(223)

           print eot, type(eot)
```
```
223 is an odd number <type 'str'>
```

DocStrings

- Essential for specifying something about the function
- Enclosed within ''' ''' or """ """
- Docstrings are different from comments

```
In [1027]: def evenOddTest(a):
               '''
               Purpose: To validate the even-ness or odd-ness of a given integer.
               Input: Variable a
               Input Type: Integer
               Output: result statement
               Output Type: String
               '''
               result = a%2   # For even values, result is zero
               # checking value equivalence with result
               if result == 0:
                   return '%d is an even number'%(a)
               else:
                   return '%d is an odd number'%(a)
```
```
In [1028]: print type(evenOddTest)
```
```
<type 'function'>
```
```
In [1029]: print evenOddTest.func_doc
```
```
    Purpose: To validate the even-ness or odd-ness of a given integer.
    Input: Variable a
    Input Type: Integer
    Output: result statement
    Output Type: String
```
```
In [1030]: print dir(evenOddTest)
```
```
['__call__', '__class__', '__closure__', '__code__', '__defaults__', '__delat
tr__', '__dict__', '__doc__', '__format__', '__get__', '__getattribute__', '_
_globals__', '__hash__', '__init__', '__module__', '__name__', '__new__', '__
reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str_
_', '__subclasshook__', 'func_closure', 'func_code', 'func_defaults', 'func_d
ict', 'func_doc', 'func_globals', 'func_name']
```

In [1031]:
```
evenOddTest.func_name
```
Out[1031]:
```
'evenOddTest'
```
In [1032]:
```
print evenOddTest.func_defaults     # prints None, as there are no default arg
uments defined
```
```
None
```

Variable Arguments (*args and **kwargs)

In [1033]:
```
#!/usr/bin/python

'''
    Purpose: Demonstration of the usage of *args and **kwargs

    *args   stores variables in tuple
    **kwargs stores variables in dictionaries

'''
def foo(firstArg, *args, **kwargs):
    print 'Necessary argument is ', firstArg
    print 'args = ', args
    print 'type(args) is', type(args)
    if args:
        for arg in args:
            print arg

    print 'kwargs = ', kwargs
    print 'type(kwargs) is', type(kwargs)
    if kwargs:
        print "The keyword arguments are\n",
        # for kwarg in kwargs:
        #    print kwarg
        for k,v in kwargs.items():
            print '%15r  --> %10r'%(k,v)
```
In [1034]:
```
foo(321)
```
```
Necessary argument is  321
args =  ()
type(args) is <type 'tuple'>
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
In [1035]: foo(1,2,3,4)
```

```
Necessary argument is  1
args =  (2, 3, 4)
type(args) is <type 'tuple'>
2
3
4
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
In [1036]: foo(99,22.0, True, [12,34, 56])
```

```
Necessary argument is  99
args =  (22.0, True, [12, 34, 56])
type(args) is <type 'tuple'>
22.0
True
[12, 34, 56]
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
In [1037]: foo(32, 56, 32, (2, [34, (2.3, 99, 0)]))
```

```
Necessary argument is  32
args =  (56, 32, (2, [34, (2.3, 99, 0)]))
type(args) is <type 'tuple'>
56
32
(2, [34, (2.3, 99, 0)])
kwargs =  {}
type(kwargs) is <type 'dict'>
```

```
In [1038]: foo(2.0, a = 1, b=2, c=3)
```

```
Necessary argument is  2.0
args =  ()
type(args) is <type 'tuple'>
kwargs =  {'a': 1, 'c': 3, 'b': 2}
type(kwargs) is <type 'dict'>
The keyword arguments are
            'a'  -->         1
            'c'  -->         3
            'b'  -->         2
```

In [1039]:
```
foo(2, 2.0, a = 1, b=2, c=3)
```

```
Necessary argument is  2
args =  (2.0,)
type(args) is <type 'tuple'>
2.0
kwargs =  {'a': 1, 'c': 3, 'b': 2}
type(kwargs) is <type 'dict'>
The keyword arguments are
            'a'  -->          1
            'c'  -->          3
            'b'  -->          2
```

In [1040]:
```
foo('a', 1, None, a = 1, b = '2', c = 3)
```

```
Necessary argument is  a
args =  (1, None)
type(args) is <type 'tuple'>
1
None
kwargs =  {'a': 1, 'c': 3, 'b': '2'}
type(kwargs) is <type 'dict'>
The keyword arguments are
            'a'  -->          1
            'c'  -->          3
            'b'  -->        '2'
```

In [1041]:
```
foo(123, courseName = 'Python', currentStatus = '40% completed', studentList =
 ['Michel', 'Naseer', 'Johson', 'Shoban'])
```

```
Necessary argument is  123
args =  ()
type(args) is <type 'tuple'>
kwargs =  {'courseName': 'Python', 'currentStatus': '40% completed', 'student
List': ['Michel', 'Naseer', 'Johson', 'Shoban']}
type(kwargs) is <type 'dict'>
The keyword arguments are
   'courseName'  -->   'Python'
'currentStatus'  --> '40% completed'
  'studentList'  --> ['Michel', 'Naseer', 'Johson', 'Shoban']
```

Lambda

In [1042]:
```
def printSquares(x):
    for i in range(x):
        print i**2,

printSquares(7)
```

```
0 1 4 9 16 25 36
```

```
In [1043]: [lambda x:x**2  for x in range(7)]

Out[1043]: [<function __main__.<lambda>>,
             <function __main__.<lambda>>,
             <function __main__.<lambda>>,
             <function __main__.<lambda>>,
             <function __main__.<lambda>>,
             <function __main__.<lambda>>,
             <function __main__.<lambda>>]
```

```
In [1044]: map(lambda x:x**2, range(7))   # map() builtin function, that takes function a
           nd collection as input

Out[1044]: [0, 1, 4, 9, 16, 25, 36]
```

```
In [1045]: filter(lambda x:x%2 == 0, range(12))   # Observe that filter() takes a conditi
           on, unlike map()

Out[1045]: [0, 2, 4, 6, 8, 10]
```

Assignment : Using the above written logic, write evenOddTest function within the map() and get the even values between 23 and 97

map()

```
In [1046]: def double(number):
               return number*2  # experession isn't returned, but the result is
```

```
In [1047]: map(double, range(45, 56))

Out[1047]: [90, 92, 94, 96, 98, 100, 102, 104, 106, 108, 110]
```

5.5 filter

```
In [1048]: def OddNessTest(number):
               return number%2 != 0
```

```
In [1049]: listOfValues = [12, 34, 56, 78, 923, 23]
```

```
In [1050]: filter(OddNessTest, listOfValues)

Out[1050]: [923, 23]
```

```
In [1051]: map(OddNessTest, listOfValues)

Out[1051]: [False, False, False, False, True, True]
```

Interview Question : Write a function to highlist the vowels in the input string

```
In [1052]: def OddNessTest(number):
               return number%2 != 0
```

```
In [1053]: listOfValues = [12, 34, 56, 78, 923, 23]
```

Interview Question : What is the difference between map() and filter()?

```
In [1054]: filter(OddNessTest, listOfValues)

Out[1054]: [923, 23]
```

```
In [1055]: map(OddNessTest, listOfValues)

Out[1055]: [False, False, False, False, True, True]
```

Interview Question : Write a function to highlist the vowels in the input string

```
In [1056]: def highlightVowels(string):
               vowels = 'ae1ouAEIOU'
               for i in string:
                   if i in vowels:
                       print i.upper(),
                   else:
                       print i.lower(),

           highlightVowels('I will give my best to achieve my goal!')

           I   w i l l   g i v E   m y   b E s t   t O   A c h i E v E   m y   g O A l !
```

```
In [1057]: def highLightVowels(string):
               newString = []
               for ch in string:
                   if ch.lower() in 'aeiou':
                       newString.append(ch.upper())
                   else:
                       newString.append(ch.lower())
               return ''.join(newString)

           print highLightVowels('python programming language')
           print highLightVowels('python programming language'.upper())

           pythOn prOgrAmmIng lAngUAgE
           pythOn prOgrAmmIng lAngUAgE
```

Assignment : Do modifications to this logic, to remove the spaces between letters; not words

zip()

- To pair the list given
- Results in list of tuples

```
In [1058]: zip('Python', 'Python')
```

```
Out[1058]: [('P', 'P'), ('y', 'y'), ('t', 't'), ('h', 'h'), ('o', 'o'), ('n', 'n')]
```

```
In [1059]: zip('python', xrange(len('python')))
```

```
Out[1059]: [('p', 0), ('y', 1), ('t', 2), ('h', 3), ('o', 4), ('n', 5)]
```

```
In [1060]: zip(xrange(len('python')), 'Python')
```

```
Out[1060]: [(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```

```
In [1061]: zip('Python')
```

```
Out[1061]: [('P',), ('y',), ('t',), ('h',), ('o',), ('n',)]
```

```
In [1062]: zip('Python', 'Python', 'python')
```

```
Out[1062]: [('P', 'P', 'p'),
            ('y', 'y', 'y'),
            ('t', 't', 't'),
            ('h', 'h', 'h'),
            ('o', 'o', 'o'),
            ('n', 'n', 'n')]
```

Interview Question : What is the difference between map() and zip()?

```
In [1063]: print map(None, range(10), range(12))    # None type function taken to pair th
           e values in list
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9,
 9), (None, 10), (None, 11)]
```

```
In [1064]: print zip(range(10), range(12))
```

```
[(0, 0), (1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6), (7, 7), (8, 8), (9,
 9)]
```

NOTE: Observe that map() can handle lists of assymmetric lengths, whereas zip() can't

reduce()

- It accepts an iterator to process, but it's not an iterator itself. It returns a single result

```
In [1065]: reduce( (lambda a, b: a * b), [1, 2, 3, 4] )   # results in 1*2*3*4

Out[1065]: 24
```

Interview Question : what is the difference between map() and reduce()?

```
In [1066]: map( (lambda a, b: a * b), [1, 2, 3, 4] )

           ---------------------------------------------------------------------------
           TypeError                                 Traceback (most recent call last)
           <ipython-input-1066-c4e63925bfeb> in <module>()
           ----> 1 map( (lambda a, b: a * b), [1, 2, 3, 4] )

           TypeError: <lambda>() takes exactly 2 arguments (1 given)

In [1067]: map( (lambda a, b: a * b), [1, 2, 3, 4], [1, 2, 3, 4] )

Out[1067]: [1, 4, 9, 16]

In [1068]: reduce( (lambda a, b: a ** b), [1, 2, 3, 4] )
                        # results in 1**2**3**4; anything to the power of 1 results in 1

Out[1068]: 1

In [1069]: map( (lambda a, b: a ** b), [1, 2, 3, 4], [1, 2, 3, 4] )

Out[1069]: [1, 4, 27, 256]

In [1070]: reduce( (lambda a, b: a ** b), [2, 3, 4] )    # results in 2**3**4

Out[1070]: 4096
```

Interview Question : Perform string concatenation using reduce()

```
In [1071]: reduce((lambda a,b:a+b), ['python', 'programming'])

Out[1071]: 'pythonprogramming'

In [1072]: reduce((lambda a,b:a + ' ' + b), ['python', 'programming'])

Out[1072]: 'python programming'

In [1073]: string = 'I am confident about my Success'
           list = string.split(' ')
           print list

           ['I', 'am', 'confident', 'about', 'my', 'Success']

In [1074]: reduce(lambda a,b: a+b, list)

Out[1074]: 'IamconfidentaboutmySuccess'
```

```
In [1075]: reduce(lambda a,b: a + ' ' + b, list)  # with spaces between words

Out[1075]: 'I am confident about my Success'
```

Assignment : Write a script with two functions, celsiusToFahrenheit() and fahrenheitToCelsius(). In runtime, the user should be given choice to enter either Fahrenheit or Celcius; then use the functions to convert correspondingly. Then, display the result from main function, using string formatting?

Assignment : Write three functions absolute(). Prompt the user to feed an integer (). The result should mimick the functionality of built-in function abs()

```
ex:
    In [1]: abs(9)
    Out[1]: 9

    In [2]: abs(-9)
    Out[2]: 9

    In [3]: abs(0)
    Out[3]: 0

    In [4]: abs(2+3j)
    Out[4]: 3.6055512754639896
```

Assignment : Write a function that validates and prints whether the run-time input given is palindrome or not.

```
Ex: 'otto'
    'Evil is a deed as I live' Hint: ignore white-space here.
```

Assignment : write a function that generates palindromes, for the given run-time input. There should be two outputs, both for even and odd case. Ref : http://www.jimsabo.com/palindrome.html (http://www.jimsabo.com/palindrome.html).

Assignment : Write a function to implement the caesar cipher, for the given input.

```
    ex1: input  --> 'Python'
        output --> 'Qzuipo'
    ex2: input -->  'One day, I will achieve my goal!'
         output ->  'Pof!ebz-!J!xjmm!bdijfwf!nz!hpbm"'
  HINT: use chr() and ord() builtin functions
```

Assignment : Define a function histogram() that takes a list of integers and prints a histogram to the screen. For example, histogram([2, 5, 7]) should print:

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

Assignment : Write a function to get the fibinocci series of numbers till 100; and another function to print the corresponding number of astericks. Hint: 0, 1, 1, 2, 3, 5, 8, ...

```
Output expected :
    *
    *
    **
    ***
    *****
    ********
    ...so on
```

Assignment : Write a function to calculate the factorial of a number.

```
Hint: factorial(4) = 4*3*2*1 = 24
```

Recursive functions

```
def funcName(<input paramaters>):
    <some logic>
    return funcName(<input parameters>)
```

Recursion is a programming technique in which a call to a function results in another call to that same function. Iteration is calling an object, and moving over it.

Problem : Return the fibonacci series (0, 1, 1, 2, 3, 5, 8) using recursions

In [1076]:
```
def fib(n):                # method 1 recursion
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1)+fib(n-2)

print fib(20)
```

```
6765
```

In [1077]:
```
fib(5)    # 5th element    # fib(4)+fib(3)
                           # fib(4) -> fib(3)+fib(2)
                           #             fib ...
```

Out[1077]:
```
5
```

In [1078]:
```
fib(6)   # 6th element
```

Out[1078]:
```
8
```

Assignment : Using recursive functions, generate the fibonacci series and display all the series still a given number

Problem : Write a recursive function to get the factorial of a given number

```
In [1079]: reduce((lambda x,y: x*y), xrange(1,12))      # method 2

Out[1079]: 39916800

In [1080]: def factorial(n):                            # method 3 recursion
               if n == 1 or 0:
                   return 1
               return n * factorial(n-1)

           factorial(12-1)

Out[1080]: 39916800

In [1081]: factorial(5) # 5*3*2*1*1

Out[1081]: 120

In [1082]: def factorial(n):       # same logic, but consolidated        # method 4 recur
           sion
               return 1 if n==0 else n*factorial(n-1)

In [1083]: factorial(12-1)

Out[1083]: 39916800

In [1084]: def factorial(n):       # same logic, but consolidated        # method 5 recur
           sion
               n = abs(n)
               return 1 if n==0 else n*factorial(n-1)

In [1085]: factorial(11)

Out[1085]: 39916800

In [1086]: factorial(-11)

Out[1086]: 39916800

In [1087]: def factorial(n):                                           # method 6 recursi
           on
               if n == 0:
                   return 1
               elif n <0:
                   return n*factorial(n+1)
               else:
                   return n*factorial(n-1)

In [1088]: factorial(0)

Out[1088]: 1
```

```
In [1089]: factorial(11)

Out[1089]: 39916800
```

```
In [1090]: factorial(-11)

Out[1090]: -39916800
```

Assignment : using timeit module, conclude which of the above factorial() functions, is faster?

Interview Question : write a recursive function to display the reverse of a given string, using recursive function.

ex:'Python' -> 'nohtyP'

```
In [1091]: def stringReverse(string):           # method 1
               if string == '':
                   return ''
               else:
                   return stringReverse(string[1:]) + string[0]

           stringReverse('Python Programming')

Out[1091]: 'gnimmargorP nohtyP'
```

```
In [1092]: def stringReverse(string):           # method 2
               if string == '':
                   return ''
               return string[-1]+stringReverse(string[:-1])

           stringReverse('Python')

Out[1092]: 'nohtyP'
```

Assignment : Modify this stringReverse() recursive function to result as below:

'Python Programming' -> 'nohtyP gnimmargorP'

NOTE: We also know that string reversal is possible using slicing (str[::-1]) and using built-in function reversed()

Problem : Write a recursive function to display the sum of squares of whole numbers, till given number, n.

```
1**2 +2**2 + 3**2 + ...
```

```
In [1093]: def squareSum(n):
               if n == 0:
                   return 0
               else:
                   return pow(n,2) + squareSum(n-1)

           squareSum(4)       #   0**2 + 1**2 + 2**2 + 3**2 + 4**2
```

```
Out[1093]: 30
```

Assignmnet : Write a recursive function to compute the sum of first 'n' whole number

Hint : 10 -> 10+9+8+ ......1+0
if n is 0, return 0

Assignment : Write a recursive function which displays the Pascal's triangle:

```
                        1
                     1    1
                   1    2    1
                1    3    3    1
             1    4    6    4    1
         1    5    10    10    5    1
   HINT: use fibonacci series
```

Assignments (Interview Question): Implement a recursive function to multiply and divide 2 numbers recursively using + and - operators only.

Assignment : Write a recursive function to display first 50 prime numbers

Assignment : WAP to display the sum of digits of a number Hint: n//10, n%10

Problem : Create an infinite loop using recursive functions

```
In [1094]: global noOfRecursions
           noOfRecursions = 0
           def loop(noOfRecursions):          # Infinite loop
               print 'Hi! I am in Loop '
               noOfRecursions+=1              # to get the count of number of recursions
            occurred
               print 'This is Loop %d'%noOfRecursions
               return loop(noOfRecursions)
```

In [13]:
```
loop(noOfRecursions)
```

```
Hi! I am in Loop
This is Loop 1
Hi! I am in Loop
This is Loop 2
Hi! I am in Loop
This is Loop 3
Hi! I am in Loop
This is Loop 4
Hi! I am in Loop
This is Loop 5
Hi! I am in Loop
This is Loop 6
Hi! I am in Loop
This is Loop 7
Hi! I am in Loop
This is Loop 8
Hi! I am in Loop
This is Loop 9
Hi! I am in Loop
This is Loop 10
Hi! I am in Loop
This is Loop 11
Hi! I am in Loop
This is Loop 12
Hi! I am in Loop
This is Loop 13
Hi! I am in Loop
This is Loop 14
Hi! I am in Loop
This is Loop 15
Hi! I am in Loop
This is Loop 16
Hi! I am in Loop
This is Loop 17
Hi! I am in Loop
This is Loop 18
Hi! I am in Loop
This is Loop 19
Hi! I am in Loop
This is Loop 20
Hi! I am in Loop
This is Loop 21
Hi! I am in Loop
This is Loop 22
Hi! I am in Loop
This is Loop 23
Hi! I am in Loop
This is Loop 24
Hi! I am in Loop
This is Loop 25
Hi! I am in Loop
This is Loop 26
Hi! I am in Loop
This is Loop 27
Hi! I am in Loop
This is Loop 28
Hi! I am in Loop
```

```
This is Loop 29
Hi! I am in Loop
This is Loop 30
Hi! I am in Loop
This is Loop 31
Hi! I am in Loop
This is Loop 32
Hi! I am in Loop
This is Loop 33
Hi! I am in Loop
This is Loop 34
Hi! I am in Loop
This is Loop 35
Hi! I am in Loop
This is Loop 36
Hi! I am in Loop
This is Loop 37
Hi! I am in Loop
This is Loop 38
Hi! I am in Loop
This is Loop 39
Hi! I am in Loop
This is Loop 40
Hi! I am in Loop
This is Loop 41
Hi! I am in Loop
This is Loop 42
Hi! I am in Loop
This is Loop 43
Hi! I am in Loop
This is Loop 44
Hi! I am in Loop
This is Loop 45
Hi! I am in Loop
This is Loop 46
Hi! I am in Loop
This is Loop 47
Hi! I am in Loop
This is Loop 48
Hi! I am in Loop
This is Loop 49
Hi! I am in Loop
This is Loop 50
Hi! I am in Loop
This is Loop 51
Hi! I am in Loop
This is Loop 52
Hi! I am in Loop
This is Loop 53
Hi! I am in Loop
This is Loop 54
Hi! I am in Loop
This is Loop 55
Hi! I am in Loop
This is Loop 56
Hi! I am in Loop
This is Loop 57
```

```
Hi! I am in Loop
This is Loop 58
Hi! I am in Loop
This is Loop 59
Hi! I am in Loop
This is Loop 60
Hi! I am in Loop
This is Loop 61
Hi! I am in Loop
This is Loop 62
Hi! I am in Loop
This is Loop 63
Hi! I am in Loop
This is Loop 64
Hi! I am in Loop
This is Loop 65
Hi! I am in Loop
This is Loop 66
Hi! I am in Loop
This is Loop 67
Hi! I am in Loop
This is Loop 68
Hi! I am in Loop
This is Loop 69
Hi! I am in Loop
This is Loop 70
Hi! I am in Loop
This is Loop 71
Hi! I am in Loop
This is Loop 72
Hi! I am in Loop
This is Loop 73
Hi! I am in Loop
This is Loop 74
Hi! I am in Loop
This is Loop 75
Hi! I am in Loop
This is Loop 76
Hi! I am in Loop
This is Loop 77
Hi! I am in Loop
This is Loop 78
Hi! I am in Loop
This is Loop 79
Hi! I am in Loop
This is Loop 80
Hi! I am in Loop
This is Loop 81
Hi! I am in Loop
This is Loop 82
Hi! I am in Loop
This is Loop 83
Hi! I am in Loop
This is Loop 84
Hi! I am in Loop
This is Loop 85
Hi! I am in Loop
```

```
This is Loop 86
Hi! I am in Loop
This is Loop 87
Hi! I am in Loop
This is Loop 88
Hi! I am in Loop
This is Loop 89
Hi! I am in Loop
This is Loop 90
Hi! I am in Loop
This is Loop 91
Hi! I am in Loop
This is Loop 92
Hi! I am in Loop
This is Loop 93
Hi! I am in Loop
This is Loop 94
Hi! I am in Loop
This is Loop 95
Hi! I am in Loop
This is Loop 96
Hi! I am in Loop
This is Loop 97
Hi! I am in Loop
This is Loop 98
Hi! I am in Loop
This is Loop 99
Hi! I am in Loop
This is Loop 100
Hi! I am in Loop
This is Loop 101
Hi! I am in Loop
This is Loop 102
Hi! I am in Loop
This is Loop 103
Hi! I am in Loop
This is Loop 104
Hi! I am in Loop
This is Loop 105
Hi! I am in Loop
This is Loop 106
Hi! I am in Loop
This is Loop 107
Hi! I am in Loop
This is Loop 108
Hi! I am in Loop
This is Loop 109
Hi! I am in Loop
This is Loop 110
Hi! I am in Loop
This is Loop 111
Hi! I am in Loop
This is Loop 112
Hi! I am in Loop
This is Loop 113
Hi! I am in Loop
This is Loop 114
```

```
Hi! I am in Loop
This is Loop 115
Hi! I am in Loop
This is Loop 116
Hi! I am in Loop
This is Loop 117
Hi! I am in Loop
This is Loop 118
Hi! I am in Loop
This is Loop 119
Hi! I am in Loop
This is Loop 120
Hi! I am in Loop
This is Loop 121
Hi! I am in Loop
This is Loop 122
Hi! I am in Loop
This is Loop 123
Hi! I am in Loop
This is Loop 124
Hi! I am in Loop
This is Loop 125
Hi! I am in Loop
This is Loop 126
Hi! I am in Loop
This is Loop 127
Hi! I am in Loop
This is Loop 128
Hi! I am in Loop
This is Loop 129
Hi! I am in Loop
This is Loop 130
Hi! I am in Loop
This is Loop 131
Hi! I am in Loop
This is Loop 132
Hi! I am in Loop
This is Loop 133
Hi! I am in Loop
This is Loop 134
Hi! I am in Loop
This is Loop 135
Hi! I am in Loop
This is Loop 136
Hi! I am in Loop
This is Loop 137
Hi! I am in Loop
This is Loop 138
Hi! I am in Loop
This is Loop 139
Hi! I am in Loop
This is Loop 140
Hi! I am in Loop
This is Loop 141
Hi! I am in Loop
This is Loop 142
Hi! I am in Loop
```

```
This is Loop 143
Hi! I am in Loop
This is Loop 144
Hi! I am in Loop
This is Loop 145
Hi! I am in Loop
This is Loop 146
Hi! I am in Loop
This is Loop 147
Hi! I am in Loop
This is Loop 148
Hi! I am in Loop
This is Loop 149
Hi! I am in Loop
This is Loop 150
Hi! I am in Loop
This is Loop 151
Hi! I am in Loop
This is Loop 152
Hi! I am in Loop
This is Loop 153
Hi! I am in Loop
This is Loop 154
Hi! I am in Loop
This is Loop 155
Hi! I am in Loop
This is Loop 156
Hi! I am in Loop
This is Loop 157
Hi! I am in Loop
This is Loop 158
Hi! I am in Loop
This is Loop 159
Hi! I am in Loop
This is Loop 160
Hi! I am in Loop
This is Loop 161
Hi! I am in Loop
This is Loop 162
Hi! I am in Loop
This is Loop 163
Hi! I am in Loop
This is Loop 164
Hi! I am in Loop
This is Loop 165
Hi! I am in Loop
This is Loop 166
Hi! I am in Loop
This is Loop 167
Hi! I am in Loop
This is Loop 168
Hi! I am in Loop
This is Loop 169
Hi! I am in Loop
This is Loop 170
Hi! I am in Loop
This is Loop 171
```

```
Hi! I am in Loop
This is Loop 172
Hi! I am in Loop
This is Loop 173
Hi! I am in Loop
This is Loop 174
Hi! I am in Loop
This is Loop 175
Hi! I am in Loop
This is Loop 176
Hi! I am in Loop
This is Loop 177
Hi! I am in Loop
This is Loop 178
Hi! I am in Loop
This is Loop 179
Hi! I am in Loop
This is Loop 180
Hi! I am in Loop
This is Loop 181
Hi! I am in Loop
This is Loop 182
Hi! I am in Loop
This is Loop 183
Hi! I am in Loop
This is Loop 184
Hi! I am in Loop
This is Loop 185
Hi! I am in Loop
This is Loop 186
Hi! I am in Loop
This is Loop 187
Hi! I am in Loop
This is Loop 188
Hi! I am in Loop
This is Loop 189
Hi! I am in Loop
This is Loop 190
Hi! I am in Loop
This is Loop 191
Hi! I am in Loop
This is Loop 192
Hi! I am in Loop
This is Loop 193
Hi! I am in Loop
This is Loop 194
Hi! I am in Loop
This is Loop 195
Hi! I am in Loop
This is Loop 196
Hi! I am in Loop
This is Loop 197
Hi! I am in Loop
This is Loop 198
Hi! I am in Loop
This is Loop 199
Hi! I am in Loop
```

```
This is Loop 200
Hi! I am in Loop
This is Loop 201
Hi! I am in Loop
This is Loop 202
Hi! I am in Loop
This is Loop 203
Hi! I am in Loop
This is Loop 204
Hi! I am in Loop
This is Loop 205
Hi! I am in Loop
This is Loop 206
Hi! I am in Loop
This is Loop 207
Hi! I am in Loop
This is Loop 208
Hi! I am in Loop
This is Loop 209
Hi! I am in Loop
This is Loop 210
Hi! I am in Loop
This is Loop 211
Hi! I am in Loop
This is Loop 212
Hi! I am in Loop
This is Loop 213
Hi! I am in Loop
This is Loop 214
Hi! I am in Loop
This is Loop 215
Hi! I am in Loop
This is Loop 216
Hi! I am in Loop
This is Loop 217
Hi! I am in Loop
This is Loop 218
Hi! I am in Loop
This is Loop 219
Hi! I am in Loop
This is Loop 220
Hi! I am in Loop
This is Loop 221
Hi! I am in Loop
This is Loop 222
Hi! I am in Loop
This is Loop 223
Hi! I am in Loop
This is Loop 224
Hi! I am in Loop
This is Loop 225
Hi! I am in Loop
This is Loop 226
Hi! I am in Loop
This is Loop 227
Hi! I am in Loop
This is Loop 228
```

```
Hi! I am in Loop
This is Loop 229
Hi! I am in Loop
This is Loop 230
Hi! I am in Loop
This is Loop 231
Hi! I am in Loop
This is Loop 232
Hi! I am in Loop
This is Loop 233
Hi! I am in Loop
This is Loop 234
Hi! I am in Loop
This is Loop 235
Hi! I am in Loop
This is Loop 236
Hi! I am in Loop
This is Loop 237
Hi! I am in Loop
This is Loop 238
Hi! I am in Loop
This is Loop 239
Hi! I am in Loop
This is Loop 240
Hi! I am in Loop
This is Loop 241
Hi! I am in Loop
This is Loop 242
Hi! I am in Loop
This is Loop 243
Hi! I am in Loop
This is Loop 244
Hi! I am in Loop
This is Loop 245
Hi! I am in Loop
This is Loop 246
Hi! I am in Loop
This is Loop 247
Hi! I am in Loop
This is Loop 248
Hi! I am in Loop
This is Loop 249
Hi! I am in Loop
This is Loop 250
Hi! I am in Loop
This is Loop 251
Hi! I am in Loop
This is Loop 252
Hi! I am in Loop
This is Loop 253
Hi! I am in Loop
This is Loop 254
Hi! I am in Loop
This is Loop 255
Hi! I am in Loop
This is Loop 256
Hi! I am in Loop
```

```
This is Loop 257
Hi! I am in Loop
This is Loop 258
Hi! I am in Loop
This is Loop 259
Hi! I am in Loop
This is Loop 260
Hi! I am in Loop
This is Loop 261
Hi! I am in Loop
This is Loop 262
Hi! I am in Loop
This is Loop 263
Hi! I am in Loop
This is Loop 264
Hi! I am in Loop
This is Loop 265
Hi! I am in Loop
This is Loop 266
Hi! I am in Loop
This is Loop 267
Hi! I am in Loop
This is Loop 268
Hi! I am in Loop
This is Loop 269
Hi! I am in Loop
This is Loop 270
Hi! I am in Loop
This is Loop 271
Hi! I am in Loop
This is Loop 272
Hi! I am in Loop
This is Loop 273
Hi! I am in Loop
This is Loop 274
Hi! I am in Loop
This is Loop 275
Hi! I am in Loop
This is Loop 276
Hi! I am in Loop
This is Loop 277
Hi! I am in Loop
This is Loop 278
Hi! I am in Loop
This is Loop 279
Hi! I am in Loop
This is Loop 280
Hi! I am in Loop
This is Loop 281
Hi! I am in Loop
This is Loop 282
Hi! I am in Loop
This is Loop 283
Hi! I am in Loop
This is Loop 284
Hi! I am in Loop
This is Loop 285
```

```
Hi! I am in Loop
This is Loop 286
Hi! I am in Loop
This is Loop 287
Hi! I am in Loop
This is Loop 288
Hi! I am in Loop
This is Loop 289
Hi! I am in Loop
This is Loop 290
Hi! I am in Loop
This is Loop 291
Hi! I am in Loop
This is Loop 292
Hi! I am in Loop
This is Loop 293
Hi! I am in Loop
This is Loop 294
Hi! I am in Loop
This is Loop 295
Hi! I am in Loop
This is Loop 296
Hi! I am in Loop
This is Loop 297
Hi! I am in Loop
This is Loop 298
Hi! I am in Loop
This is Loop 299
Hi! I am in Loop
This is Loop 300
Hi! I am in Loop
This is Loop 301
Hi! I am in Loop
This is Loop 302
Hi! I am in Loop
This is Loop 303
Hi! I am in Loop
This is Loop 304
Hi! I am in Loop
This is Loop 305
Hi! I am in Loop
This is Loop 306
Hi! I am in Loop
This is Loop 307
Hi! I am in Loop
This is Loop 308
Hi! I am in Loop
This is Loop 309
Hi! I am in Loop
This is Loop 310
Hi! I am in Loop
This is Loop 311
Hi! I am in Loop
This is Loop 312
Hi! I am in Loop
This is Loop 313
Hi! I am in Loop
```

```
This is Loop 314
Hi! I am in Loop
This is Loop 315
Hi! I am in Loop
This is Loop 316
Hi! I am in Loop
This is Loop 317
Hi! I am in Loop
This is Loop 318
Hi! I am in Loop
This is Loop 319
Hi! I am in Loop
This is Loop 320
Hi! I am in Loop
This is Loop 321
Hi! I am in Loop
This is Loop 322
Hi! I am in Loop
This is Loop 323
Hi! I am in Loop
This is Loop 324
Hi! I am in Loop
This is Loop 325
Hi! I am in Loop
This is Loop 326
Hi! I am in Loop
This is Loop 327
Hi! I am in Loop
This is Loop 328
Hi! I am in Loop
This is Loop 329
Hi! I am in Loop
This is Loop 330
Hi! I am in Loop
This is Loop 331
Hi! I am in Loop
This is Loop 332
Hi! I am in Loop
This is Loop 333
Hi! I am in Loop
This is Loop 334
Hi! I am in Loop
This is Loop 335
Hi! I am in Loop
This is Loop 336
Hi! I am in Loop
This is Loop 337
Hi! I am in Loop
This is Loop 338
Hi! I am in Loop
This is Loop 339
Hi! I am in Loop
This is Loop 340
Hi! I am in Loop
This is Loop 341
Hi! I am in Loop
This is Loop 342
```

```
Hi! I am in Loop
This is Loop 343
Hi! I am in Loop
This is Loop 344
Hi! I am in Loop
This is Loop 345
Hi! I am in Loop
This is Loop 346
Hi! I am in Loop
This is Loop 347
Hi! I am in Loop
This is Loop 348
Hi! I am in Loop
This is Loop 349
Hi! I am in Loop
This is Loop 350
Hi! I am in Loop
This is Loop 351
Hi! I am in Loop
This is Loop 352
Hi! I am in Loop
This is Loop 353
Hi! I am in Loop
This is Loop 354
Hi! I am in Loop
This is Loop 355
Hi! I am in Loop
This is Loop 356
Hi! I am in Loop
This is Loop 357
Hi! I am in Loop
This is Loop 358
Hi! I am in Loop
This is Loop 359
Hi! I am in Loop
This is Loop 360
Hi! I am in Loop
This is Loop 361
Hi! I am in Loop
This is Loop 362
Hi! I am in Loop
This is Loop 363
Hi! I am in Loop
This is Loop 364
Hi! I am in Loop
This is Loop 365
Hi! I am in Loop
This is Loop 366
Hi! I am in Loop
This is Loop 367
Hi! I am in Loop
This is Loop 368
Hi! I am in Loop
This is Loop 369
Hi! I am in Loop
This is Loop 370
Hi! I am in Loop
```

```
This is Loop 371
Hi! I am in Loop
This is Loop 372
Hi! I am in Loop
This is Loop 373
Hi! I am in Loop
This is Loop 374
Hi! I am in Loop
This is Loop 375
Hi! I am in Loop
This is Loop 376
Hi! I am in Loop
This is Loop 377
Hi! I am in Loop
This is Loop 378
Hi! I am in Loop
This is Loop 379
Hi! I am in Loop
This is Loop 380
Hi! I am in Loop
This is Loop 381
Hi! I am in Loop
This is Loop 382
Hi! I am in Loop
This is Loop 383
Hi! I am in Loop
This is Loop 384
Hi! I am in Loop
This is Loop 385
Hi! I am in Loop
This is Loop 386
Hi! I am in Loop
This is Loop 387
Hi! I am in Loop
This is Loop 388
Hi! I am in Loop
This is Loop 389
Hi! I am in Loop
This is Loop 390
Hi! I am in Loop
This is Loop 391
Hi! I am in Loop
This is Loop 392
Hi! I am in Loop
This is Loop 393
Hi! I am in Loop
This is Loop 394
Hi! I am in Loop
This is Loop 395
Hi! I am in Loop
This is Loop 396
Hi! I am in Loop
This is Loop 397
Hi! I am in Loop
This is Loop 398
Hi! I am in Loop
This is Loop 399
```

```
Hi! I am in Loop
This is Loop 400
Hi! I am in Loop
This is Loop 401
Hi! I am in Loop
This is Loop 402
Hi! I am in Loop
This is Loop 403
Hi! I am in Loop
This is Loop 404
Hi! I am in Loop
This is Loop 405
Hi! I am in Loop
This is Loop 406
Hi! I am in Loop
This is Loop 407
Hi! I am in Loop
This is Loop 408
Hi! I am in Loop
This is Loop 409
Hi! I am in Loop
This is Loop 410
Hi! I am in Loop
This is Loop 411
Hi! I am in Loop
This is Loop 412
Hi! I am in Loop
This is Loop 413
Hi! I am in Loop
This is Loop 414
Hi! I am in Loop
This is Loop 415
Hi! I am in Loop
This is Loop 416
Hi! I am in Loop
This is Loop 417
Hi! I am in Loop
This is Loop 418
Hi! I am in Loop
This is Loop 419
Hi! I am in Loop
This is Loop 420
Hi! I am in Loop
This is Loop 421
Hi! I am in Loop
This is Loop 422
Hi! I am in Loop
This is Loop 423
Hi! I am in Loop
This is Loop 424
Hi! I am in Loop
This is Loop 425
Hi! I am in Loop
This is Loop 426
Hi! I am in Loop
This is Loop 427
Hi! I am in Loop
```

```
This is Loop 428
Hi! I am in Loop
This is Loop 429
Hi! I am in Loop
This is Loop 430
Hi! I am in Loop
This is Loop 431
Hi! I am in Loop
This is Loop 432
Hi! I am in Loop
This is Loop 433
Hi! I am in Loop
This is Loop 434
Hi! I am in Loop
This is Loop 435
Hi! I am in Loop
This is Loop 436
Hi! I am in Loop
This is Loop 437
Hi! I am in Loop
This is Loop 438
Hi! I am in Loop
This is Loop 439
Hi! I am in Loop
This is Loop 440
Hi! I am in Loop
This is Loop 441
Hi! I am in Loop
This is Loop 442
Hi! I am in Loop
This is Loop 443
Hi! I am in Loop
This is Loop 444
Hi! I am in Loop
This is Loop 445
Hi! I am in Loop
This is Loop 446
Hi! I am in Loop
This is Loop 447
Hi! I am in Loop
This is Loop 448
Hi! I am in Loop
This is Loop 449
Hi! I am in Loop
This is Loop 450
Hi! I am in Loop
This is Loop 451
Hi! I am in Loop
This is Loop 452
Hi! I am in Loop
This is Loop 453
Hi! I am in Loop
This is Loop 454
Hi! I am in Loop
This is Loop 455
Hi! I am in Loop
This is Loop 456
```

```
Hi! I am in Loop
This is Loop 457
Hi! I am in Loop
This is Loop 458
Hi! I am in Loop
This is Loop 459
Hi! I am in Loop
This is Loop 460
Hi! I am in Loop
This is Loop 461
Hi! I am in Loop
This is Loop 462
Hi! I am in Loop
This is Loop 463
Hi! I am in Loop
This is Loop 464
Hi! I am in Loop
This is Loop 465
Hi! I am in Loop
This is Loop 466
Hi! I am in Loop
This is Loop 467
Hi! I am in Loop
This is Loop 468
Hi! I am in Loop
This is Loop 469
Hi! I am in Loop
This is Loop 470
Hi! I am in Loop
This is Loop 471
Hi! I am in Loop
This is Loop 472
Hi! I am in Loop
This is Loop 473
Hi! I am in Loop
This is Loop 474
Hi! I am in Loop
This is Loop 475
Hi! I am in Loop
This is Loop 476
Hi! I am in Loop
This is Loop 477
Hi! I am in Loop
This is Loop 478
Hi! I am in Loop
This is Loop 479
Hi! I am in Loop
This is Loop 480
Hi! I am in Loop
This is Loop 481
Hi! I am in Loop
This is Loop 482
Hi! I am in Loop
This is Loop 483
Hi! I am in Loop
This is Loop 484
Hi! I am in Loop
```

```
This is Loop 485
Hi! I am in Loop
This is Loop 486
Hi! I am in Loop
This is Loop 487
Hi! I am in Loop
This is Loop 488
Hi! I am in Loop
This is Loop 489
Hi! I am in Loop
This is Loop 490
Hi! I am in Loop
This is Loop 491
Hi! I am in Loop
This is Loop 492
Hi! I am in Loop
This is Loop 493
Hi! I am in Loop
This is Loop 494
Hi! I am in Loop
This is Loop 495
Hi! I am in Loop
This is Loop 496
Hi! I am in Loop
This is Loop 497
Hi! I am in Loop
This is Loop 498
Hi! I am in Loop
This is Loop 499
Hi! I am in Loop
This is Loop 500
Hi! I am in Loop
This is Loop 501
Hi! I am in Loop
This is Loop 502
Hi! I am in Loop
This is Loop 503
Hi! I am in Loop
This is Loop 504
Hi! I am in Loop
This is Loop 505
Hi! I am in Loop
This is Loop 506
Hi! I am in Loop
This is Loop 507
Hi! I am in Loop
This is Loop 508
Hi! I am in Loop
This is Loop 509
Hi! I am in Loop
This is Loop 510
Hi! I am in Loop
This is Loop 511
Hi! I am in Loop
This is Loop 512
Hi! I am in Loop
This is Loop 513
```

```
Hi! I am in Loop
This is Loop 514
Hi! I am in Loop
This is Loop 515
Hi! I am in Loop
This is Loop 516
Hi! I am in Loop
This is Loop 517
Hi! I am in Loop
This is Loop 518
Hi! I am in Loop
This is Loop 519
Hi! I am in Loop
This is Loop 520
Hi! I am in Loop
This is Loop 521
Hi! I am in Loop
This is Loop 522
Hi! I am in Loop
This is Loop 523
Hi! I am in Loop
This is Loop 524
Hi! I am in Loop
This is Loop 525
Hi! I am in Loop
This is Loop 526
Hi! I am in Loop
This is Loop 527
Hi! I am in Loop
This is Loop 528
Hi! I am in Loop
This is Loop 529
Hi! I am in Loop
This is Loop 530
Hi! I am in Loop
This is Loop 531
Hi! I am in Loop
This is Loop 532
Hi! I am in Loop
This is Loop 533
Hi! I am in Loop
This is Loop 534
Hi! I am in Loop
This is Loop 535
Hi! I am in Loop
This is Loop 536
Hi! I am in Loop
This is Loop 537
Hi! I am in Loop
This is Loop 538
Hi! I am in Loop
This is Loop 539
Hi! I am in Loop
This is Loop 540
Hi! I am in Loop
This is Loop 541
Hi! I am in Loop
```

```
This is Loop 542
Hi! I am in Loop
This is Loop 543
Hi! I am in Loop
This is Loop 544
Hi! I am in Loop
This is Loop 545
Hi! I am in Loop
This is Loop 546
Hi! I am in Loop
This is Loop 547
Hi! I am in Loop
This is Loop 548
Hi! I am in Loop
This is Loop 549
Hi! I am in Loop
This is Loop 550
Hi! I am in Loop
This is Loop 551
Hi! I am in Loop
This is Loop 552
Hi! I am in Loop
This is Loop 553
Hi! I am in Loop
This is Loop 554
Hi! I am in Loop
This is Loop 555
Hi! I am in Loop
This is Loop 556
Hi! I am in Loop
This is Loop 557
Hi! I am in Loop
This is Loop 558
Hi! I am in Loop
This is Loop 559
Hi! I am in Loop
This is Loop 560
Hi! I am in Loop
This is Loop 561
Hi! I am in Loop
This is Loop 562
Hi! I am in Loop
This is Loop 563
Hi! I am in Loop
This is Loop 564
Hi! I am in Loop
This is Loop 565
Hi! I am in Loop
This is Loop 566
Hi! I am in Loop
This is Loop 567
Hi! I am in Loop
This is Loop 568
Hi! I am in Loop
This is Loop 569
Hi! I am in Loop
This is Loop 570
```

```
Hi! I am in Loop
This is Loop 571
Hi! I am in Loop
This is Loop 572
Hi! I am in Loop
This is Loop 573
Hi! I am in Loop
This is Loop 574
Hi! I am in Loop
This is Loop 575
Hi! I am in Loop
This is Loop 576
Hi! I am in Loop
This is Loop 577
Hi! I am in Loop
This is Loop 578
Hi! I am in Loop
This is Loop 579
Hi! I am in Loop
This is Loop 580
Hi! I am in Loop
This is Loop 581
Hi! I am in Loop
This is Loop 582
Hi! I am in Loop
This is Loop 583
Hi! I am in Loop
This is Loop 584
Hi! I am in Loop
This is Loop 585
Hi! I am in Loop
This is Loop 586
Hi! I am in Loop
This is Loop 587
Hi! I am in Loop
This is Loop 588
Hi! I am in Loop
This is Loop 589
Hi! I am in Loop
This is Loop 590
Hi! I am in Loop
This is Loop 591
Hi! I am in Loop
This is Loop 592
Hi! I am in Loop
This is Loop 593
Hi! I am in Loop
This is Loop 594
Hi! I am in Loop
This is Loop 595
Hi! I am in Loop
This is Loop 596
Hi! I am in Loop
This is Loop 597
Hi! I am in Loop
This is Loop 598
Hi! I am in Loop
```

```
This is Loop 599
Hi! I am in Loop
This is Loop 600
Hi! I am in Loop
This is Loop 601
Hi! I am in Loop
This is Loop 602
Hi! I am in Loop
This is Loop 603
Hi! I am in Loop
This is Loop 604
Hi! I am in Loop
This is Loop 605
Hi! I am in Loop
This is Loop 606
Hi! I am in Loop
This is Loop 607
Hi! I am in Loop
This is Loop 608
Hi! I am in Loop
This is Loop 609
Hi! I am in Loop
This is Loop 610
Hi! I am in Loop
This is Loop 611
Hi! I am in Loop
This is Loop 612
Hi! I am in Loop
This is Loop 613
Hi! I am in Loop
This is Loop 614
Hi! I am in Loop
This is Loop 615
Hi! I am in Loop
This is Loop 616
Hi! I am in Loop
This is Loop 617
Hi! I am in Loop
This is Loop 618
Hi! I am in Loop
This is Loop 619
Hi! I am in Loop
This is Loop 620
Hi! I am in Loop
This is Loop 621
Hi! I am in Loop
This is Loop 622
Hi! I am in Loop
This is Loop 623
Hi! I am in Loop
This is Loop 624
Hi! I am in Loop
This is Loop 625
Hi! I am in Loop
This is Loop 626
Hi! I am in Loop
This is Loop 627
```

```
Hi! I am in Loop
This is Loop 628
Hi! I am in Loop
This is Loop 629
Hi! I am in Loop
This is Loop 630
Hi! I am in Loop
This is Loop 631
Hi! I am in Loop
This is Loop 632
Hi! I am in Loop
This is Loop 633
Hi! I am in Loop
This is Loop 634
Hi! I am in Loop
This is Loop 635
Hi! I am in Loop
This is Loop 636
Hi! I am in Loop
This is Loop 637
Hi! I am in Loop
This is Loop 638
Hi! I am in Loop
This is Loop 639
Hi! I am in Loop
This is Loop 640
Hi! I am in Loop
This is Loop 641
Hi! I am in Loop
This is Loop 642
Hi! I am in Loop
This is Loop 643
Hi! I am in Loop
This is Loop 644
Hi! I am in Loop
This is Loop 645
Hi! I am in Loop
This is Loop 646
Hi! I am in Loop
This is Loop 647
Hi! I am in Loop
This is Loop 648
Hi! I am in Loop
This is Loop 649
Hi! I am in Loop
This is Loop 650
Hi! I am in Loop
This is Loop 651
Hi! I am in Loop
This is Loop 652
Hi! I am in Loop
This is Loop 653
Hi! I am in Loop
This is Loop 654
Hi! I am in Loop
This is Loop 655
Hi! I am in Loop
```

```
This is Loop 656
Hi! I am in Loop
This is Loop 657
Hi! I am in Loop
This is Loop 658
Hi! I am in Loop
This is Loop 659
Hi! I am in Loop
This is Loop 660
Hi! I am in Loop
This is Loop 661
Hi! I am in Loop
This is Loop 662
Hi! I am in Loop
This is Loop 663
Hi! I am in Loop
This is Loop 664
Hi! I am in Loop
This is Loop 665
Hi! I am in Loop
This is Loop 666
Hi! I am in Loop
This is Loop 667
Hi! I am in Loop
This is Loop 668
Hi! I am in Loop
This is Loop 669
Hi! I am in Loop
This is Loop 670
Hi! I am in Loop
This is Loop 671
Hi! I am in Loop
This is Loop 672
Hi! I am in Loop
This is Loop 673
Hi! I am in Loop
This is Loop 674
Hi! I am in Loop
This is Loop 675
Hi! I am in Loop
This is Loop 676
Hi! I am in Loop
This is Loop 677
Hi! I am in Loop
This is Loop 678
Hi! I am in Loop
This is Loop 679
Hi! I am in Loop
This is Loop 680
Hi! I am in Loop
This is Loop 681
Hi! I am in Loop
This is Loop 682
Hi! I am in Loop
This is Loop 683
Hi! I am in Loop
This is Loop 684
```

```
Hi! I am in Loop
This is Loop 685
Hi! I am in Loop
This is Loop 686
Hi! I am in Loop
This is Loop 687
Hi! I am in Loop
This is Loop 688
Hi! I am in Loop
This is Loop 689
Hi! I am in Loop
This is Loop 690
Hi! I am in Loop
This is Loop 691
Hi! I am in Loop
This is Loop 692
Hi! I am in Loop
This is Loop 693
Hi! I am in Loop
This is Loop 694
Hi! I am in Loop
This is Loop 695
Hi! I am in Loop
This is Loop 696
Hi! I am in Loop
This is Loop 697
Hi! I am in Loop
This is Loop 698
Hi! I am in Loop
This is Loop 699
Hi! I am in Loop
This is Loop 700
Hi! I am in Loop
This is Loop 701
Hi! I am in Loop
This is Loop 702
Hi! I am in Loop
This is Loop 703
Hi! I am in Loop
This is Loop 704
Hi! I am in Loop
This is Loop 705
Hi! I am in Loop
This is Loop 706
Hi! I am in Loop
This is Loop 707
Hi! I am in Loop
This is Loop 708
Hi! I am in Loop
This is Loop 709
Hi! I am in Loop
This is Loop 710
Hi! I am in Loop
This is Loop 711
Hi! I am in Loop
This is Loop 712
Hi! I am in Loop
```

```
This is Loop 713
Hi! I am in Loop
This is Loop 714
Hi! I am in Loop
This is Loop 715
Hi! I am in Loop
This is Loop 716
Hi! I am in Loop
This is Loop 717
Hi! I am in Loop
This is Loop 718
Hi! I am in Loop
This is Loop 719
Hi! I am in Loop
This is Loop 720
Hi! I am in Loop
This is Loop 721
Hi! I am in Loop
This is Loop 722
Hi! I am in Loop
This is Loop 723
Hi! I am in Loop
This is Loop 724
Hi! I am in Loop
This is Loop 725
Hi! I am in Loop
This is Loop 726
Hi! I am in Loop
This is Loop 727
Hi! I am in Loop
This is Loop 728
Hi! I am in Loop
This is Loop 729
Hi! I am in Loop
This is Loop 730
Hi! I am in Loop
This is Loop 731
Hi! I am in Loop
This is Loop 732
Hi! I am in Loop
This is Loop 733
Hi! I am in Loop
This is Loop 734
Hi! I am in Loop
This is Loop 735
Hi! I am in Loop
This is Loop 736
Hi! I am in Loop
This is Loop 737
Hi! I am in Loop
This is Loop 738
Hi! I am in Loop
This is Loop 739
Hi! I am in Loop
This is Loop 740
Hi! I am in Loop
This is Loop 741
```

```
Hi! I am in Loop
This is Loop 742
Hi! I am in Loop
This is Loop 743
Hi! I am in Loop
This is Loop 744
Hi! I am in Loop
This is Loop 745
Hi! I am in Loop
This is Loop 746
Hi! I am in Loop
This is Loop 747
Hi! I am in Loop
This is Loop 748
Hi! I am in Loop
This is Loop 749
Hi! I am in Loop
This is Loop 750
Hi! I am in Loop
This is Loop 751
Hi! I am in Loop
This is Loop 752
Hi! I am in Loop
This is Loop 753
Hi! I am in Loop
This is Loop 754
Hi! I am in Loop
This is Loop 755
Hi! I am in Loop
This is Loop 756
Hi! I am in Loop
This is Loop 757
Hi! I am in Loop
This is Loop 758
Hi! I am in Loop
This is Loop 759
Hi! I am in Loop
This is Loop 760
Hi! I am in Loop
This is Loop 761
Hi! I am in Loop
This is Loop 762
Hi! I am in Loop
This is Loop 763
Hi! I am in Loop
This is Loop 764
Hi! I am in Loop
This is Loop 765
Hi! I am in Loop
This is Loop 766
Hi! I am in Loop
This is Loop 767
Hi! I am in Loop
This is Loop 768
Hi! I am in Loop
This is Loop 769
Hi! I am in Loop
```

```
This is Loop 770
Hi! I am in Loop
This is Loop 771
Hi! I am in Loop
This is Loop 772
Hi! I am in Loop
This is Loop 773
Hi! I am in Loop
This is Loop 774
Hi! I am in Loop
This is Loop 775
Hi! I am in Loop
This is Loop 776
Hi! I am in Loop
This is Loop 777
Hi! I am in Loop
This is Loop 778
Hi! I am in Loop
This is Loop 779
Hi! I am in Loop
This is Loop 780
Hi! I am in Loop
This is Loop 781
Hi! I am in Loop
This is Loop 782
Hi! I am in Loop
This is Loop 783
Hi! I am in Loop
This is Loop 784
Hi! I am in Loop
This is Loop 785
Hi! I am in Loop
This is Loop 786
Hi! I am in Loop
This is Loop 787
Hi! I am in Loop
This is Loop 788
Hi! I am in Loop
This is Loop 789
Hi! I am in Loop
This is Loop 790
Hi! I am in Loop
This is Loop 791
Hi! I am in Loop
This is Loop 792
Hi! I am in Loop
This is Loop 793
Hi! I am in Loop
This is Loop 794
Hi! I am in Loop
This is Loop 795
Hi! I am in Loop
This is Loop 796
Hi! I am in Loop
This is Loop 797
Hi! I am in Loop
This is Loop 798
```

```
Hi! I am in Loop
This is Loop 799
Hi! I am in Loop
This is Loop 800
Hi! I am in Loop
This is Loop 801
Hi! I am in Loop
This is Loop 802
Hi! I am in Loop
This is Loop 803
Hi! I am in Loop
This is Loop 804
Hi! I am in Loop
This is Loop 805
Hi! I am in Loop
This is Loop 806
Hi! I am in Loop
This is Loop 807
Hi! I am in Loop
This is Loop 808
Hi! I am in Loop
This is Loop 809
Hi! I am in Loop
This is Loop 810
Hi! I am in Loop
This is Loop 811
Hi! I am in Loop
This is Loop 812
Hi! I am in Loop
This is Loop 813
Hi! I am in Loop
This is Loop 814
Hi! I am in Loop
This is Loop 815
Hi! I am in Loop
This is Loop 816
Hi! I am in Loop
This is Loop 817
Hi! I am in Loop
This is Loop 818
Hi! I am in Loop
This is Loop 819
Hi! I am in Loop
This is Loop 820
Hi! I am in Loop
This is Loop 821
Hi! I am in Loop
This is Loop 822
Hi! I am in Loop
This is Loop 823
Hi! I am in Loop
This is Loop 824
Hi! I am in Loop
This is Loop 825
Hi! I am in Loop
This is Loop 826
Hi! I am in Loop
```

```
This is Loop 827
Hi! I am in Loop
This is Loop 828
Hi! I am in Loop
This is Loop 829
Hi! I am in Loop
This is Loop 830
Hi! I am in Loop
This is Loop 831
Hi! I am in Loop
This is Loop 832
Hi! I am in Loop
This is Loop 833
Hi! I am in Loop
This is Loop 834
Hi! I am in Loop
This is Loop 835
Hi! I am in Loop
This is Loop 836
Hi! I am in Loop
This is Loop 837
Hi! I am in Loop
This is Loop 838
Hi! I am in Loop
This is Loop 839
Hi! I am in Loop
This is Loop 840
Hi! I am in Loop
This is Loop 841
Hi! I am in Loop
This is Loop 842
Hi! I am in Loop
This is Loop 843
Hi! I am in Loop
This is Loop 844
Hi! I am in Loop
This is Loop 845
Hi! I am in Loop
This is Loop 846
Hi! I am in Loop
This is Loop 847
Hi! I am in Loop
This is Loop 848
Hi! I am in Loop
This is Loop 849
Hi! I am in Loop
This is Loop 850
Hi! I am in Loop
This is Loop 851
Hi! I am in Loop
This is Loop 852
Hi! I am in Loop
This is Loop 853
Hi! I am in Loop
This is Loop 854
Hi! I am in Loop
This is Loop 855
```

```
Hi! I am in Loop
This is Loop 856
Hi! I am in Loop
This is Loop 857
Hi! I am in Loop
This is Loop 858
Hi! I am in Loop
This is Loop 859
Hi! I am in Loop
This is Loop 860
Hi! I am in Loop
This is Loop 861
Hi! I am in Loop
This is Loop 862
Hi! I am in Loop
This is Loop 863
Hi! I am in Loop
This is Loop 864
Hi! I am in Loop
This is Loop 865
Hi! I am in Loop
This is Loop 866
Hi! I am in Loop
This is Loop 867
Hi! I am in Loop
This is Loop 868
Hi! I am in Loop
This is Loop 869
Hi! I am in Loop
This is Loop 870
Hi! I am in Loop
This is Loop 871
Hi! I am in Loop
This is Loop 872
Hi! I am in Loop
This is Loop 873
Hi! I am in Loop
This is Loop 874
Hi! I am in Loop
This is Loop 875
Hi! I am in Loop
This is Loop 876
Hi! I am in Loop
This is Loop 877
Hi! I am in Loop
This is Loop 878
Hi! I am in Loop
This is Loop 879
Hi! I am in Loop
This is Loop 880
Hi! I am in Loop
This is Loop 881
Hi! I am in Loop
This is Loop 882
Hi! I am in Loop
This is Loop 883
Hi! I am in Loop
```

```
This is Loop 884
Hi! I am in Loop
This is Loop 885
Hi! I am in Loop
This is Loop 886
Hi! I am in Loop
This is Loop 887
Hi! I am in Loop
This is Loop 888
Hi! I am in Loop
This is Loop 889
Hi! I am in Loop
This is Loop 890
Hi! I am in Loop
This is Loop 891
Hi! I am in Loop
This is Loop 892
Hi! I am in Loop
This is Loop 893
Hi! I am in Loop
This is Loop 894
Hi! I am in Loop
This is Loop 895
Hi! I am in Loop
This is Loop 896
Hi! I am in Loop
This is Loop 897
Hi! I am in Loop
This is Loop 898
Hi! I am in Loop
This is Loop 899
Hi! I am in Loop
This is Loop 900
Hi! I am in Loop
This is Loop 901
Hi! I am in Loop
This is Loop 902
Hi! I am in Loop
This is Loop 903
Hi! I am in Loop
This is Loop 904
Hi! I am in Loop
This is Loop 905
Hi! I am in Loop
This is Loop 906
Hi! I am in Loop
This is Loop 907
Hi! I am in Loop
This is Loop 908
Hi! I am in Loop
This is Loop 909
Hi! I am in Loop
This is Loop 910
Hi! I am in Loop
This is Loop 911
Hi! I am in Loop
This is Loop 912
```

```
Hi! I am in Loop
This is Loop 913
Hi! I am in Loop
This is Loop 914
Hi! I am in Loop
This is Loop 915
Hi! I am in Loop
This is Loop 916
Hi! I am in Loop
This is Loop 917
Hi! I am in Loop
This is Loop 918
Hi! I am in Loop
This is Loop 919
Hi! I am in Loop
This is Loop 920
Hi! I am in Loop
This is Loop 921
Hi! I am in Loop
This is Loop 922
Hi! I am in Loop
This is Loop 923
Hi! I am in Loop
This is Loop 924
Hi! I am in Loop
This is Loop 925
Hi! I am in Loop
This is Loop 926
Hi! I am in Loop
This is Loop 927
Hi! I am in Loop
This is Loop 928
Hi! I am in Loop
This is Loop 929
Hi! I am in Loop
This is Loop 930
Hi! I am in Loop
This is Loop 931
Hi! I am in Loop
This is Loop 932
Hi! I am in Loop
This is Loop 933
Hi! I am in Loop
This is Loop 934
Hi! I am in Loop
This is Loop 935
Hi! I am in Loop
This is Loop 936
Hi! I am in Loop
This is Loop 937
Hi! I am in Loop
This is Loop 938
Hi! I am in Loop
This is Loop 939
Hi! I am in Loop
This is Loop 940
Hi! I am in Loop
```

```
This is Loop 941
Hi! I am in Loop
This is Loop 942
Hi! I am in Loop
This is Loop 943
Hi! I am in Loop
This is Loop 944
Hi! I am in Loop
This is Loop 945
Hi! I am in Loop
This is Loop 946
Hi! I am in Loop
This is Loop 947
Hi! I am in Loop
This is Loop 948
Hi! I am in Loop
This is Loop 949
Hi! I am in Loop
This is Loop 950
Hi! I am in Loop
This is Loop 951
Hi! I am in Loop
This is Loop 952
Hi! I am in Loop
This is Loop 953
Hi! I am in Loop
This is Loop 954
Hi! I am in Loop
This is Loop 955
Hi! I am in Loop
This is Loop 956
Hi! I am in Loop
This is Loop 957
Hi! I am in Loop
This is Loop 958
Hi! I am in Loop
This is Loop 959
Hi! I am in Loop
This is Loop 960
Hi! I am in Loop
This is Loop 961
Hi! I am in Loop
This is Loop 962
Hi! I am in Loop
This is Loop 963
Hi! I am in Loop
This is Loop 964
Hi! I am in Loop
This is Loop 965
Hi! I am in Loop
This is Loop 966
Hi! I am in Loop
This is Loop 967
Hi! I am in Loop
This is Loop 968
Hi! I am in Loop
This is Loop 969
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-13-479a6f6d0cf4> in <module>()
----> 1 loop(noOfRecursions)

<ipython-input-12-06673717ce8c> in loop(noOfRecursions)
      6     noOfRecursions+=1              # to get the count of number of re
cursions occurred
      7     print 'This is Loop %d'%noOfRecursions
----> 8     return loop(noOfRecursions)

... last 1 frames repeated, from the frame below ...

<ipython-input-12-06673717ce8c> in loop(noOfRecursions)
      6     noOfRecursions+=1              # to get the count of number of re
cursions occurred
      7     print 'This is Loop %d'%noOfRecursions
----> 8     return loop(noOfRecursions)

RuntimeError: maximum recursion depth exceeded while calling a Python object
```

In [1095]:
```
print noOfRecursions   # It results the default value given before entering th
e function, as returns from subframes was halted.
```

```
0
```

Notice that in python, infinite loop will not run for infinite time. But, they gets halted after reaching the maximum recursion depth.

Interview question : what is the maximum recursion depth of any python object? Does it differ for different types of objects? Is it dependent on the machine being used?

Problem : Write a function to demonstrate the MUTUAL recursion between two functions

In [1096]:
```
# Infinite loop between these functions  --- Mutual recursion
global noOfloops
noOfloops = 0
def func1():
    print 'I am in function 1 .'
    global noOfloops
    noOfloops+=1
    print noOfloops
    return func2()

def func2():
    print 'I am in function 2 .'
    global noOfloops
    noOfloops+=1
    print noOfloops
    return func1()
```

In [47]:
```
func1()
```

```
I am in function 1 .
1
I am in function 2 .
2
I am in function 1 .
3
I am in function 2 .
4
I am in function 1 .
5
I am in function 2 .
6
I am in function 1 .
7
I am in function 2 .
8
I am in function 1 .
9
I am in function 2 .
10
I am in function 1 .
11
I am in function 2 .
12
I am in function 1 .
13
I am in function 2 .
14
I am in function 1 .
15
I am in function 2 .
16
I am in function 1 .
17
I am in function 2 .
18
I am in function 1 .
19
I am in function 2 .
20
I am in function 1 .
21
I am in function 2 .
22
I am in function 1 .
23
I am in function 2 .
24
I am in function 1 .
25
I am in function 2 .
26
I am in function 1 .
27
I am in function 2 .
28
I am in function 1 .
```

```
29
I am in function 2 .
30
I am in function 1 .
31
I am in function 2 .
32
I am in function 1 .
33
I am in function 2 .
34
I am in function 1 .
35
I am in function 2 .
36
I am in function 1 .
37
I am in function 2 .
38
I am in function 1 .
39
I am in function 2 .
40
I am in function 1 .
41
I am in function 2 .
42
I am in function 1 .
43
I am in function 2 .
44
I am in function 1 .
45
I am in function 2 .
46
I am in function 1 .
47
I am in function 2 .
48
I am in function 1 .
49
I am in function 2 .
50
I am in function 1 .
51
I am in function 2 .
52
I am in function 1 .
53
I am in function 2 .
54
I am in function 1 .
55
I am in function 2 .
56
I am in function 1 .
57
```

```
I am in function 2 .
58
I am in function 1 .
59
I am in function 2 .
60
I am in function 1 .
61
I am in function 2 .
62
I am in function 1 .
63
I am in function 2 .
64
I am in function 1 .
65
I am in function 2 .
66
I am in function 1 .
67
I am in function 2 .
68
I am in function 1 .
69
I am in function 2 .
70
I am in function 1 .
71
I am in function 2 .
72
I am in function 1 .
73
I am in function 2 .
74
I am in function 1 .
75
I am in function 2 .
76
I am in function 1 .
77
I am in function 2 .
78
I am in function 1 .
79
I am in function 2 .
80
I am in function 1 .
81
I am in function 2 .
82
I am in function 1 .
83
I am in function 2 .
84
I am in function 1 .
85
I am in function 2 .
```

```
86
I am in function 1 .
87
I am in function 2 .
88
I am in function 1 .
89
I am in function 2 .
90
I am in function 1 .
91
I am in function 2 .
92
I am in function 1 .
93
I am in function 2 .
94
I am in function 1 .
95
I am in function 2 .
96
I am in function 1 .
97
I am in function 2 .
98
I am in function 1 .
99
I am in function 2 .
100
I am in function 1 .
101
I am in function 2 .
102
I am in function 1 .
103
I am in function 2 .
104
I am in function 1 .
105
I am in function 2 .
106
I am in function 1 .
107
I am in function 2 .
108
I am in function 1 .
109
I am in function 2 .
110
I am in function 1 .
111
I am in function 2 .
112
I am in function 1 .
113
I am in function 2 .
114
```

```
I am in function 1 .
115
I am in function 2 .
116
I am in function 1 .
117
I am in function 2 .
118
I am in function 1 .
119
I am in function 2 .
120
I am in function 1 .
121
I am in function 2 .
122
I am in function 1 .
123
I am in function 2 .
124
I am in function 1 .
125
I am in function 2 .
126
I am in function 1 .
127
I am in function 2 .
128
I am in function 1 .
129
I am in function 2 .
130
I am in function 1 .
131
I am in function 2 .
132
I am in function 1 .
133
I am in function 2 .
134
I am in function 1 .
135
I am in function 2 .
136
I am in function 1 .
137
I am in function 2 .
138
I am in function 1 .
139
I am in function 2 .
140
I am in function 1 .
141
I am in function 2 .
142
I am in function 1 .
```

```
143
I am in function 2 .
144
I am in function 1 .
145
I am in function 2 .
146
I am in function 1 .
147
I am in function 2 .
148
I am in function 1 .
149
I am in function 2 .
150
I am in function 1 .
151
I am in function 2 .
152
I am in function 1 .
153
I am in function 2 .
154
I am in function 1 .
155
I am in function 2 .
156
I am in function 1 .
157
I am in function 2 .
158
I am in function 1 .
159
I am in function 2 .
160
I am in function 1 .
161
I am in function 2 .
162
I am in function 1 .
163
I am in function 2 .
164
I am in function 1 .
165
I am in function 2 .
166
I am in function 1 .
167
I am in function 2 .
168
I am in function 1 .
169
I am in function 2 .
170
I am in function 1 .
171
```

```
I am in function 2 .
172
I am in function 1 .
173
I am in function 2 .
174
I am in function 1 .
175
I am in function 2 .
176
I am in function 1 .
177
I am in function 2 .
178
I am in function 1 .
179
I am in function 2 .
180
I am in function 1 .
181
I am in function 2 .
182
I am in function 1 .
183
I am in function 2 .
184
I am in function 1 .
185
I am in function 2 .
186
I am in function 1 .
187
I am in function 2 .
188
I am in function 1 .
189
I am in function 2 .
190
I am in function 1 .
191
I am in function 2 .
192
I am in function 1 .
193
I am in function 2 .
194
I am in function 1 .
195
I am in function 2 .
196
I am in function 1 .
197
I am in function 2 .
198
I am in function 1 .
199
I am in function 2 .
```

```
200
I am in function 1 .
201
I am in function 2 .
202
I am in function 1 .
203
I am in function 2 .
204
I am in function 1 .
205
I am in function 2 .
206
I am in function 1 .
207
I am in function 2 .
208
I am in function 1 .
209
I am in function 2 .
210
I am in function 1 .
211
I am in function 2 .
212
I am in function 1 .
213
I am in function 2 .
214
I am in function 1 .
215
I am in function 2 .
216
I am in function 1 .
217
I am in function 2 .
218
I am in function 1 .
219
I am in function 2 .
220
I am in function 1 .
221
I am in function 2 .
222
I am in function 1 .
223
I am in function 2 .
224
I am in function 1 .
225
I am in function 2 .
226
I am in function 1 .
227
I am in function 2 .
228
```

```
I am in function 1 .
229
I am in function 2 .
230
I am in function 1 .
231
I am in function 2 .
232
I am in function 1 .
233
I am in function 2 .
234
I am in function 1 .
235
I am in function 2 .
236
I am in function 1 .
237
I am in function 2 .
238
I am in function 1 .
239
I am in function 2 .
240
I am in function 1 .
241
I am in function 2 .
242
I am in function 1 .
243
I am in function 2 .
244
I am in function 1 .
245
I am in function 2 .
246
I am in function 1 .
247
I am in function 2 .
248
I am in function 1 .
249
I am in function 2 .
250
I am in function 1 .
251
I am in function 2 .
252
I am in function 1 .
253
I am in function 2 .
254
I am in function 1 .
255
I am in function 2 .
256
I am in function 1 .
```

```
257
I am in function 2 .
258
I am in function 1 .
259
I am in function 2 .
260
I am in function 1 .
261
I am in function 2 .
262
I am in function 1 .
263
I am in function 2 .
264
I am in function 1 .
265
I am in function 2 .
266
I am in function 1 .
267
I am in function 2 .
268
I am in function 1 .
269
I am in function 2 .
270
I am in function 1 .
271
I am in function 2 .
272
I am in function 1 .
273
I am in function 2 .
274
I am in function 1 .
275
I am in function 2 .
276
I am in function 1 .
277
I am in function 2 .
278
I am in function 1 .
279
I am in function 2 .
280
I am in function 1 .
281
I am in function 2 .
282
I am in function 1 .
283
I am in function 2 .
284
I am in function 1 .
285
```

```
I am in function 2 .
286
I am in function 1 .
287
I am in function 2 .
288
I am in function 1 .
289
I am in function 2 .
290
I am in function 1 .
291
I am in function 2 .
292
I am in function 1 .
293
I am in function 2 .
294
I am in function 1 .
295
I am in function 2 .
296
I am in function 1 .
297
I am in function 2 .
298
I am in function 1 .
299
I am in function 2 .
300
I am in function 1 .
301
I am in function 2 .
302
I am in function 1 .
303
I am in function 2 .
304
I am in function 1 .
305
I am in function 2 .
306
I am in function 1 .
307
I am in function 2 .
308
I am in function 1 .
309
I am in function 2 .
310
I am in function 1 .
311
I am in function 2 .
312
I am in function 1 .
313
I am in function 2 .
```

```
314
I am in function 1 .
315
I am in function 2 .
316
I am in function 1 .
317
I am in function 2 .
318
I am in function 1 .
319
I am in function 2 .
320
I am in function 1 .
321
I am in function 2 .
322
I am in function 1 .
323
I am in function 2 .
324
I am in function 1 .
325
I am in function 2 .
326
I am in function 1 .
327
I am in function 2 .
328
I am in function 1 .
329
I am in function 2 .
330
I am in function 1 .
331
I am in function 2 .
332
I am in function 1 .
333
I am in function 2 .
334
I am in function 1 .
335
I am in function 2 .
336
I am in function 1 .
337
I am in function 2 .
338
I am in function 1 .
339
I am in function 2 .
340
I am in function 1 .
341
I am in function 2 .
342
```

```
I am in function 1 .
343
I am in function 2 .
344
I am in function 1 .
345
I am in function 2 .
346
I am in function 1 .
347
I am in function 2 .
348
I am in function 1 .
349
I am in function 2 .
350
I am in function 1 .
351
I am in function 2 .
352
I am in function 1 .
353
I am in function 2 .
354
I am in function 1 .
355
I am in function 2 .
356
I am in function 1 .
357
I am in function 2 .
358
I am in function 1 .
359
I am in function 2 .
360
I am in function 1 .
361
I am in function 2 .
362
I am in function 1 .
363
I am in function 2 .
364
I am in function 1 .
365
I am in function 2 .
366
I am in function 1 .
367
I am in function 2 .
368
I am in function 1 .
369
I am in function 2 .
370
I am in function 1 .
```

```
371
I am in function 2 .
372
I am in function 1 .
373
I am in function 2 .
374
I am in function 1 .
375
I am in function 2 .
376
I am in function 1 .
377
I am in function 2 .
378
I am in function 1 .
379
I am in function 2 .
380
I am in function 1 .
381
I am in function 2 .
382
I am in function 1 .
383
I am in function 2 .
384
I am in function 1 .
385
I am in function 2 .
386
I am in function 1 .
387
I am in function 2 .
388
I am in function 1 .
389
I am in function 2 .
390
I am in function 1 .
391
I am in function 2 .
392
I am in function 1 .
393
I am in function 2 .
394
I am in function 1 .
395
I am in function 2 .
396
I am in function 1 .
397
I am in function 2 .
398
I am in function 1 .
399
```

```
I am in function 2 .
400
I am in function 1 .
401
I am in function 2 .
402
I am in function 1 .
403
I am in function 2 .
404
I am in function 1 .
405
I am in function 2 .
406
I am in function 1 .
407
I am in function 2 .
408
I am in function 1 .
409
I am in function 2 .
410
I am in function 1 .
411
I am in function 2 .
412
I am in function 1 .
413
I am in function 2 .
414
I am in function 1 .
415
I am in function 2 .
416
I am in function 1 .
417
I am in function 2 .
418
I am in function 1 .
419
I am in function 2 .
420
I am in function 1 .
421
I am in function 2 .
422
I am in function 1 .
423
I am in function 2 .
424
I am in function 1 .
425
I am in function 2 .
426
I am in function 1 .
427
I am in function 2 .
```

```
428
I am in function 1 .
429
I am in function 2 .
430
I am in function 1 .
431
I am in function 2 .
432
I am in function 1 .
433
I am in function 2 .
434
I am in function 1 .
435
I am in function 2 .
436
I am in function 1 .
437
I am in function 2 .
438
I am in function 1 .
439
I am in function 2 .
440
I am in function 1 .
441
I am in function 2 .
442
I am in function 1 .
443
I am in function 2 .
444
I am in function 1 .
445
I am in function 2 .
446
I am in function 1 .
447
I am in function 2 .
448
I am in function 1 .
449
I am in function 2 .
450
I am in function 1 .
451
I am in function 2 .
452
I am in function 1 .
453
I am in function 2 .
454
I am in function 1 .
455
I am in function 2 .
456
```

```
I am in function 1 .
457
I am in function 2 .
458
I am in function 1 .
459
I am in function 2 .
460
I am in function 1 .
461
I am in function 2 .
462
I am in function 1 .
463
I am in function 2 .
464
I am in function 1 .
465
I am in function 2 .
466
I am in function 1 .
467
I am in function 2 .
468
I am in function 1 .
469
I am in function 2 .
470
I am in function 1 .
471
I am in function 2 .
472
I am in function 1 .
473
I am in function 2 .
474
I am in function 1 .
475
I am in function 2 .
476
I am in function 1 .
477
I am in function 2 .
478
I am in function 1 .
479
I am in function 2 .
480
I am in function 1 .
481
I am in function 2 .
482
I am in function 1 .
483
I am in function 2 .
484
I am in function 1 .
```

```
485
I am in function 2 .
486
I am in function 1 .
487
I am in function 2 .
488
I am in function 1 .
489
I am in function 2 .
490
I am in function 1 .
491
I am in function 2 .
492
I am in function 1 .
493
I am in function 2 .
494
I am in function 1 .
495
I am in function 2 .
496
I am in function 1 .
497
I am in function 2 .
498
I am in function 1 .
499
I am in function 2 .
500
I am in function 1 .
501
I am in function 2 .
502
I am in function 1 .
503
I am in function 2 .
504
I am in function 1 .
505
I am in function 2 .
506
I am in function 1 .
507
I am in function 2 .
508
I am in function 1 .
509
I am in function 2 .
510
I am in function 1 .
511
I am in function 2 .
512
I am in function 1 .
513
```

```
I am in function 2 .
514
I am in function 1 .
515
I am in function 2 .
516
I am in function 1 .
517
I am in function 2 .
518
I am in function 1 .
519
I am in function 2 .
520
I am in function 1 .
521
I am in function 2 .
522
I am in function 1 .
523
I am in function 2 .
524
I am in function 1 .
525
I am in function 2 .
526
I am in function 1 .
527
I am in function 2 .
528
I am in function 1 .
529
I am in function 2 .
530
I am in function 1 .
531
I am in function 2 .
532
I am in function 1 .
533
I am in function 2 .
534
I am in function 1 .
535
I am in function 2 .
536
I am in function 1 .
537
I am in function 2 .
538
I am in function 1 .
539
I am in function 2 .
540
I am in function 1 .
541
I am in function 2 .
```

```
542
I am in function 1 .
543
I am in function 2 .
544
I am in function 1 .
545
I am in function 2 .
546
I am in function 1 .
547
I am in function 2 .
548
I am in function 1 .
549
I am in function 2 .
550
I am in function 1 .
551
I am in function 2 .
552
I am in function 1 .
553
I am in function 2 .
554
I am in function 1 .
555
I am in function 2 .
556
I am in function 1 .
557
I am in function 2 .
558
I am in function 1 .
559
I am in function 2 .
560
I am in function 1 .
561
I am in function 2 .
562
I am in function 1 .
563
I am in function 2 .
564
I am in function 1 .
565
I am in function 2 .
566
I am in function 1 .
567
I am in function 2 .
568
I am in function 1 .
569
I am in function 2 .
570
```

```
I am in function 1 .
571
I am in function 2 .
572
I am in function 1 .
573
I am in function 2 .
574
I am in function 1 .
575
I am in function 2 .
576
I am in function 1 .
577
I am in function 2 .
578
I am in function 1 .
579
I am in function 2 .
580
I am in function 1 .
581
I am in function 2 .
582
I am in function 1 .
583
I am in function 2 .
584
I am in function 1 .
585
I am in function 2 .
586
I am in function 1 .
587
I am in function 2 .
588
I am in function 1 .
589
I am in function 2 .
590
I am in function 1 .
591
I am in function 2 .
592
I am in function 1 .
593
I am in function 2 .
594
I am in function 1 .
595
I am in function 2 .
596
I am in function 1 .
597
I am in function 2 .
598
I am in function 1 .
```

```
599
I am in function 2 .
600
I am in function 1 .
601
I am in function 2 .
602
I am in function 1 .
603
I am in function 2 .
604
I am in function 1 .
605
I am in function 2 .
606
I am in function 1 .
607
I am in function 2 .
608
I am in function 1 .
609
I am in function 2 .
610
I am in function 1 .
611
I am in function 2 .
612
I am in function 1 .
613
I am in function 2 .
614
I am in function 1 .
615
I am in function 2 .
616
I am in function 1 .
617
I am in function 2 .
618
I am in function 1 .
619
I am in function 2 .
620
I am in function 1 .
621
I am in function 2 .
622
I am in function 1 .
623
I am in function 2 .
624
I am in function 1 .
625
I am in function 2 .
626
I am in function 1 .
627
```

```
I am in function 2 .
628
I am in function 1 .
629
I am in function 2 .
630
I am in function 1 .
631
I am in function 2 .
632
I am in function 1 .
633
I am in function 2 .
634
I am in function 1 .
635
I am in function 2 .
636
I am in function 1 .
637
I am in function 2 .
638
I am in function 1 .
639
I am in function 2 .
640
I am in function 1 .
641
I am in function 2 .
642
I am in function 1 .
643
I am in function 2 .
644
I am in function 1 .
645
I am in function 2 .
646
I am in function 1 .
647
I am in function 2 .
648
I am in function 1 .
649
I am in function 2 .
650
I am in function 1 .
651
I am in function 2 .
652
I am in function 1 .
653
I am in function 2 .
654
I am in function 1 .
655
I am in function 2 .
```

```
656
I am in function 1 .
657
I am in function 2 .
658
I am in function 1 .
659
I am in function 2 .
660
I am in function 1 .
661
I am in function 2 .
662
I am in function 1 .
663
I am in function 2 .
664
I am in function 1 .
665
I am in function 2 .
666
I am in function 1 .
667
I am in function 2 .
668
I am in function 1 .
669
I am in function 2 .
670
I am in function 1 .
671
I am in function 2 .
672
I am in function 1 .
673
I am in function 2 .
674
I am in function 1 .
675
I am in function 2 .
676
I am in function 1 .
677
I am in function 2 .
678
I am in function 1 .
679
I am in function 2 .
680
I am in function 1 .
681
I am in function 2 .
682
I am in function 1 .
683
I am in function 2 .
684
```

```
I am in function 1 .
685
I am in function 2 .
686
I am in function 1 .
687
I am in function 2 .
688
I am in function 1 .
689
I am in function 2 .
690
I am in function 1 .
691
I am in function 2 .
692
I am in function 1 .
693
I am in function 2 .
694
I am in function 1 .
695
I am in function 2 .
696
I am in function 1 .
697
I am in function 2 .
698
I am in function 1 .
699
I am in function 2 .
700
I am in function 1 .
701
I am in function 2 .
702
I am in function 1 .
703
I am in function 2 .
704
I am in function 1 .
705
I am in function 2 .
706
I am in function 1 .
707
I am in function 2 .
708
I am in function 1 .
709
I am in function 2 .
710
I am in function 1 .
711
I am in function 2 .
712
I am in function 1 .
```

```
713
I am in function 2 .
714
I am in function 1 .
715
I am in function 2 .
716
I am in function 1 .
717
I am in function 2 .
718
I am in function 1 .
719
I am in function 2 .
720
I am in function 1 .
721
I am in function 2 .
722
I am in function 1 .
723
I am in function 2 .
724
I am in function 1 .
725
I am in function 2 .
726
I am in function 1 .
727
I am in function 2 .
728
I am in function 1 .
729
I am in function 2 .
730
I am in function 1 .
731
I am in function 2 .
732
I am in function 1 .
733
I am in function 2 .
734
I am in function 1 .
735
I am in function 2 .
736
I am in function 1 .
737
I am in function 2 .
738
I am in function 1 .
739
I am in function 2 .
740
I am in function 1 .
741
```

```
I am in function 2 .
742
I am in function 1 .
743
I am in function 2 .
744
I am in function 1 .
745
I am in function 2 .
746
I am in function 1 .
747
I am in function 2 .
748
I am in function 1 .
749
I am in function 2 .
750
I am in function 1 .
751
I am in function 2 .
752
I am in function 1 .
753
I am in function 2 .
754
I am in function 1 .
755
I am in function 2 .
756
I am in function 1 .
757
I am in function 2 .
758
I am in function 1 .
759
I am in function 2 .
760
I am in function 1 .
761
I am in function 2 .
762
I am in function 1 .
763
I am in function 2 .
764
I am in function 1 .
765
I am in function 2 .
766
I am in function 1 .
767
I am in function 2 .
768
I am in function 1 .
769
I am in function 2 .
```

```
770
I am in function 1 .
771
I am in function 2 .
772
I am in function 1 .
773
I am in function 2 .
774
I am in function 1 .
775
I am in function 2 .
776
I am in function 1 .
777
I am in function 2 .
778
I am in function 1 .
779
I am in function 2 .
780
I am in function 1 .
781
I am in function 2 .
782
I am in function 1 .
783
I am in function 2 .
784
I am in function 1 .
785
I am in function 2 .
786
I am in function 1 .
787
I am in function 2 .
788
I am in function 1 .
789
I am in function 2 .
790
I am in function 1 .
791
I am in function 2 .
792
I am in function 1 .
793
I am in function 2 .
794
I am in function 1 .
795
I am in function 2 .
796
I am in function 1 .
797
I am in function 2 .
798
```

```
I am in function 1 .
799
I am in function 2 .
800
I am in function 1 .
801
I am in function 2 .
802
I am in function 1 .
803
I am in function 2 .
804
I am in function 1 .
805
I am in function 2 .
806
I am in function 1 .
807
I am in function 2 .
808
I am in function 1 .
809
I am in function 2 .
810
I am in function 1 .
811
I am in function 2 .
812
I am in function 1 .
813
I am in function 2 .
814
I am in function 1 .
815
I am in function 2 .
816
I am in function 1 .
817
I am in function 2 .
818
I am in function 1 .
819
I am in function 2 .
820
I am in function 1 .
821
I am in function 2 .
822
I am in function 1 .
823
I am in function 2 .
824
I am in function 1 .
825
I am in function 2 .
826
I am in function 1 .
```

```
827
I am in function 2 .
828
I am in function 1 .
829
I am in function 2 .
830
I am in function 1 .
831
I am in function 2 .
832
I am in function 1 .
833
I am in function 2 .
834
I am in function 1 .
835
I am in function 2 .
836
I am in function 1 .
837
I am in function 2 .
838
I am in function 1 .
839
I am in function 2 .
840
I am in function 1 .
841
I am in function 2 .
842
I am in function 1 .
843
I am in function 2 .
844
I am in function 1 .
845
I am in function 2 .
846
I am in function 1 .
847
I am in function 2 .
848
I am in function 1 .
849
I am in function 2 .
850
I am in function 1 .
851
I am in function 2 .
852
I am in function 1 .
853
I am in function 2 .
854
I am in function 1 .
855
```

```
I am in function 2 .
856
I am in function 1 .
857
I am in function 2 .
858
I am in function 1 .
859
I am in function 2 .
860
I am in function 1 .
861
I am in function 2 .
862
I am in function 1 .
863
I am in function 2 .
864
I am in function 1 .
865
I am in function 2 .
866
I am in function 1 .
867
I am in function 2 .
868
I am in function 1 .
869
I am in function 2 .
870
I am in function 1 .
871
I am in function 2 .
872
I am in function 1 .
873
I am in function 2 .
874
I am in function 1 .
875
I am in function 2 .
876
I am in function 1 .
877
I am in function 2 .
878
I am in function 1 .
879
I am in function 2 .
880
I am in function 1 .
881
I am in function 2 .
882
I am in function 1 .
883
I am in function 2 .
```

```
884
I am in function 1 .
885
I am in function 2 .
886
I am in function 1 .
887
I am in function 2 .
888
I am in function 1 .
889
I am in function 2 .
890
I am in function 1 .
891
I am in function 2 .
892
I am in function 1 .
893
I am in function 2 .
894
I am in function 1 .
895
I am in function 2 .
896
I am in function 1 .
897
I am in function 2 .
898
I am in function 1 .
899
I am in function 2 .
900
I am in function 1 .
901
I am in function 2 .
902
I am in function 1 .
903
I am in function 2 .
904
I am in function 1 .
905
I am in function 2 .
906
I am in function 1 .
907
I am in function 2 .
908
I am in function 1 .
909
I am in function 2 .
910
I am in function 1 .
911
I am in function 2 .
912
```

```
I am in function 1 .
913
I am in function 2 .
914
I am in function 1 .
915
I am in function 2 .
916
I am in function 1 .
917
I am in function 2 .
918
I am in function 1 .
919
I am in function 2 .
920
I am in function 1 .
921
I am in function 2 .
922
I am in function 1 .
923
I am in function 2 .
924
I am in function 1 .
925
I am in function 2 .
926
I am in function 1 .
927
I am in function 2 .
928
I am in function 1 .
929
I am in function 2 .
930
I am in function 1 .
931
I am in function 2 .
932
I am in function 1 .
933
I am in function 2 .
934
I am in function 1 .
935
I am in function 2 .
936
I am in function 1 .
937
I am in function 2 .
938
I am in function 1 .
939
I am in function 2 .
940
I am in function 1 .
```

```
941
I am in function 2 .
942
I am in function 1 .
943
I am in function 2 .
944
I am in function 1 .
945
I am in function 2 .
946
I am in function 1 .
947
I am in function 2 .
948
I am in function 1 .
949
I am in function 2 .
950
I am in function 1 .
951
I am in function 2 .
952
I am in function 1 .
953
I am in function 2 .
954
I am in function 1 .
955
I am in function 2 .
956
I am in function 1 .
957
I am in function 2 .
958
I am in function 1 .
959
I am in function 2 .
960
I am in function 1 .
961
I am in function 2 .
962
I am in function 1 .
963
I am in function 2 .
964
I am in function 1 .
965
I am in function 2 .
966
I am in function 1 .
967
I am in function 2 .
968
I am in function 1 .
969
```

```
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-47-043607b7b3c9> in <module>()
----> 1 func1()

<ipython-input-46-fec7b5a3b174> in func1()
      7     noOfloops+=1
      8     print noOfloops
----> 9     return func2()
     10 
     11 def func2():

<ipython-input-46-fec7b5a3b174> in func2()
     14     noOfloops+=1
     15     print noOfloops
---> 16     return func1()

... last 2 frames repeated, from the frame below ...

<ipython-input-46-fec7b5a3b174> in func1()
      7     noOfloops+=1
      8     print noOfloops
----> 9     return func2()
     10 
     11 def func2():

RuntimeError: maximum recursion depth exceeded while calling a Python object
```

NOTE :

- Python doesnot have Tail Call optimization(TCO), to handle the recursive functions.
- It is very difficult to add TCO to python, as it is a dynamic language.

Assignment : Execute the below Script in a new .py file, and observe the result

```
#!/usr/bin/python
# listDirsNfilesUsingRecursions.py
# Purpose: To list the directories and files, in the given location
import os

def get_dirlist(path):
    """
      Return a sorted list of all entries in path.
      This returns just the names, not the full path to the names.
    """
    dirlist = os.listdir(path)
    dirlist.sort()
    return dirlist

def print_files(path, prefix = ""):
    """ Print recursive listing of contents of path """
    if prefix == "":  # Detect outermost call, print a heading
        print("Folder listing for", path)
        prefix = "| "

    dirlist = get_dirlist(path)
    for f in dirlist:
        print(prefix+f)                    # Print the line
        fullname = os.path.join(path, f)   # Turn name into full pathname
        if os.path.isdir(fullname):        # If a directory, recurse.
            print_files(fullname, prefix + "| ")

import sys
if sys.platform == 'win32':
    d = raw_input('Enter a dircetory path: ')
    print_files(repr(d))                   # r'C:\Python27\Tools'
else:
    d = raw_input('Enter a dircetory path: ')
    print_files(repr(d))
```

In **conclusion**, note that recursive calls are expensive (inefficient) as they take up a lot of memory and time. Recursive functions are hard to debug.

Recommended References:

1. Animated explanation of recursions (http://www.composingprograms.com/pages/17-recursive-functions.html)

In [1097]:
```python
#!/usr/bin/python
# Switch case equivalent implementation in python
# switchCaseEquivalent.py
case = {'a':'apple', 'b': 'banana', 'c': 'cat'}

caseChoice = raw_input('Enter the case choice(a, b or c only):')

if caseChoice in case:
    if caseChoice == 'a':
        print case.get('a', None)
    elif caseChoice == 'b':
        print case.get('b', None)
    else:
        print case.get('c', None)
else:
    print "Please enter a valid case choice: a, b or c"
```

```
Enter the case choice(a, b or c only):b
banana
```

Assignment : In switchCaseEquivalent.py, try to add a choice for reattempt, to the user.

6.0 Modules

- Both buitin (ex: os), installed (ex: django) or user-defined
- It is a collection of functions, to serve a particular purpose
- imported in the functions, using 'import'
- Not all modules will be part of basic distribution
- To install a new module, **pip install (moduleName)**
- To search a module, **pip search (moduleName)**

In [1098]:
```python
import sys
```

In [1099]:
```python
print dir(sys)
```

```
['__displayhook__', '__doc__', '__excepthook__', '__name__', '__package__',
 '__stderr__', '__stdin__', '__stdout__', '_clear_type_cache', '_current_fram
es', '_getframe', '_mercurial', 'api_version', 'argv', 'builtin_module_name
s', 'byteorder', 'call_tracing', 'callstats', 'copyright', 'displayhook', 'dl
lhandle', 'dont_write_bytecode', 'exc_clear', 'exc_info', 'exc_type', 'except
hook', 'exec_prefix', 'executable', 'exit', 'exitfunc', 'flags', 'float_inf
o', 'float_repr_style', 'getcheckinterval', 'getdefaultencoding', 'getfilesys
temencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',
 'gettrace', 'getwindowsversion', 'hexversion', 'last_traceback', 'last_typ
e', 'last_value', 'long_info', 'maxint', 'maxsize', 'maxunicode', 'meta_pat
h', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'pref
ix', 'ps1', 'ps2', 'ps3', 'py3kwarning', 'setcheckinterval', 'setprofile', 's
etrecursionlimit', 'settrace', 'stderr', 'stdin', 'stdout', 'subversion', 've
rsion', 'version_info', 'warnoptions', 'winver']
```

```
In [1100]: sys.version

Out[1100]: '2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Int
           el)]'
```

```
In [1101]: sys.version_info

Out[1101]: sys.version_info(major=2, minor=7, micro=12, releaselevel='final', serial=0)
```

```
In [1102]: sys.winver

Out[1102]: '2.7'
```

```
In [1103]: sys.path

Out[1103]: ['',
            'C:\\Windows\\system32\\python27.zip',
            'c:\\python27\\DLLs',
            'c:\\python27\\lib',
            'c:\\python27\\lib\\plat-win',
            'c:\\python27\\lib\\lib-tk',
            'c:\\python27',
            'c:\\python27\\lib\\site-packages',
            'c:\\python27\\lib\\site-packages\\win32',
            'c:\\python27\\lib\\site-packages\\win32\\lib',
            'c:\\python27\\lib\\site-packages\\Pythonwin',
            'c:\\python27\\lib\\site-packages\\IPython\\extensions',
            'C:\\Users\\PRAUD01\\.ipython']
```

Assignment : Display the sys.path from interpreter and a script file separately and notice the difference in the outputs

```
In [1104]: sys.platform

Out[1104]: 'win32'
```

```
In [1105]: sys.getwindowsversion

Out[1105]: <function sys.getwindowsversion>
```

```
In [1106]: callable(sys.getwindowsversion)  # callable() builtin function to get whether
            a particular object is callable or not

Out[1106]: True
```

```
In [1107]: sys.getwindowsversion()

Out[1107]: sys.getwindowsversion(major=6, minor=1, build=7601, platform=2, service_pack
           ='Service Pack 1')
```

```
In [1108]: callable(sys.maxint)

Out[1108]: False
```

```
In [1109]: sys.maxint
Out[1109]: 2147483647
```

```
In [1110]: sys.maxsize
Out[1110]: 2147483647
```

NOTE: User-defined modules are prioritized to builtin(or installed) modules

```
In [1111]: help(sys)   #help(sys.winver)
```

```
Help on built-in module sys:

NAME
    sys

FILE
    (built-in)

MODULE DOCS
    http://docs.python.org/library/sys

DESCRIPTION
    This module provides access to some objects used or maintained by the
    interpreter and to functions that interact strongly with the interpreter.

    Dynamic objects:

    argv -- command line arguments; argv[0] is the script pathname if known
    path -- module search path; path[0] is the script directory, else ''
    modules -- dictionary of loaded modules

    displayhook -- called to show results in an interactive session
    excepthook -- called to handle any uncaught exception other than SystemEx
it
      To customize printing in an interactive session or to install a custom
      top-level exception handler, assign other functions to replace these.

    exitfunc -- if sys.exitfunc exists, this routine is called when Python ex
its
      Assigning to sys.exitfunc is deprecated; use the atexit module instead.

    stdin -- standard input file object; used by raw_input() and input()
    stdout -- standard output file object; used by the print statement
    stderr -- standard error object; used for error messages
      By assigning other file objects (or objects that behave like files)
      to these, it is possible to redirect all of the interpreter's I/O.

    last_type -- type of last uncaught exception
    last_value -- value of last uncaught exception
    last_traceback -- traceback of last uncaught exception
      These three are only available in an interactive session after a
      traceback has been printed.

    exc_type -- type of exception currently being handled
    exc_value -- value of exception currently being handled
    exc_traceback -- traceback of exception currently being handled
      The function exc_info() should be used instead of these three,
      because it is thread-safe.

    Static objects:

    float_info -- a dict with information about the float inplementation.
    long_info -- a struct sequence with information about the long implementa
tion.
    maxint -- the largest supported integer (the smallest is -maxint-1)
    maxsize -- the largest supported length of containers.
    maxunicode -- the largest supported character
    builtin_module_names -- tuple of module names built into this interpreter
    version -- the version of this interpreter as a string
    version_info -- version information as a named tuple
    hexversion -- version information encoded as a single integer
    copyright -- copyright notice pertaining to this interpreter
    platform -- platform identifier
    executable -- absolute path of the executable binary of the Python interp
reter
    prefix -- prefix used to find the Python library
    exec_prefix -- prefix used to find the machine-specific Python library
    float_repr_style -- string indicating the style of repr() output for floa
ts
    dllhandle -- [Windows only] integer handle of the Python DLL
    winver -- [Windows only] version number of the Python DLL
    __stdin__ -- the original stdin; don't touch!
    __stdout__ -- the original stdout; don't touch!
    __stderr__ -- the original stderr; don't touch!
    __displayhook__ -- the original displayhook; don't touch!
    __excepthook__ -- the original excepthook; don't touch!

    Functions:

    displayhook() -- print an object to the screen, and save it in __builtin_
_._
    excepthook() -- print an exception and its traceback to sys.stderr
    exc_info() -- return thread-safe information about the current exception
    exc_clear() -- clear the exception state for the current thread
    exit() -- exit the interpreter by raising SystemExit
    getdlopenflags() -- returns flags to be used for dlopen() calls
    getprofile() -- get the global profiling function
    getrefcount() -- return the reference count for an object (plus one :-)
    getrecursionlimit() -- return the max recursion depth for the interpreter
    getsizeof() -- return the size of an object in bytes
    gettrace() -- get the global debug tracing function
    setcheckinterval() -- control how often the interpreter checks for events
    setdlopenflags() -- set the flags to be used for dlopen() calls
    setprofile() -- set the global profiling function
    setrecursionlimit() -- set the max recursion depth for the interpreter
    settrace() -- set the global debug tracing function

FUNCTIONS
    __displayhook__ = displayhook(...)
        displayhook(object) -> None

        Print an object to sys.stdout and also save it in __builtin__._

    __excepthook__ = excepthook(...)
        excepthook(exctype, value, traceback) -> None

        Handle an exception by displaying it with a traceback on sys.stderr.

    call_tracing(...)
        call_tracing(func, args) -> object

        Call func(*args), while tracing is enabled.  The tracing state is
        saved, and restored afterwards.  This is intended to be called from
        a debugger from a checkpoint, to recursively debug some other code.
    callstats(...)
        callstats() -> tuple of integers
        
        Return a tuple of function call statistics, if CALL_PROFILE was defin
ed
        when Python was built.  Otherwise, return None.
        
        When enabled, this function returns detailed, implementation-specific
        details about the number of function calls executed. The return value
 is
        a 11-tuple where the entries in the tuple are counts of:
        0. all function calls
        1. calls to PyFunction_Type objects
        2. PyFunction calls that do not create an argument tuple
        3. PyFunction calls that do not create an argument tuple
           and bypass PyEval_EvalCodeEx()
        4. PyMethod calls
        5. PyMethod calls on bound methods
        6. PyType calls
        7. PyCFunction calls
        8. generator calls
        9. All other calls
        10. Number of stack pops performed by call_function()
    
    exc_clear(...)
        exc_clear() -> None
        
        Clear global information on the current exception.  Subsequent calls
 to
        exc_info() will return (None,None,None) until another exception is ra
ised
        in the current thread or the execution stack returns to a frame where
        another exception is being handled.
    
    exc_info(...)
        exc_info() -> (type, value, traceback)
        
        Return information about the most recent exception caught by an excep
t
        clause in the current stack frame or in an older stack frame.
    
    exit(...)
        exit([status])
        
        Exit the interpreter by raising SystemExit(status).
        If the status is omitted or None, it defaults to zero (i.e., succes
s).
        If the status is an integer, it will be used as the system exit statu
s.
        If it is another kind of object, it will be printed and the system
        exit status will be one (i.e., failure).
    
    getcheckinterval(...)
        getcheckinterval() -> current check interval; see setcheckinterval().
    
    getdefaultencoding(...)
```

```
            getdefaultencoding() -> string
            
            Return the current default string encoding used by the Unicode
            implementation.
        
        getfilesystemencoding(...)
            getfilesystemencoding() -> string
            
            Return the encoding used to convert Unicode filenames in
            operating system filenames.
        
        getprofile(...)
            getprofile()
            
            Return the profiling function set with sys.setprofile.
            See the profiler chapter in the library manual.
        
        getrecursionlimit(...)
            getrecursionlimit()
            
            Return the current value of the recursion limit, the maximum depth
            of the Python interpreter stack.  This limit prevents infinite
            recursion from causing an overflow of the C stack and crashing Pytho
n.
        
        getrefcount(...)
            getrefcount(object) -> integer
            
            Return the reference count of object.  The count returned is generall
y
            one higher than you might expect, because it includes the (temporary)
            reference as an argument to getrefcount().
        
        getsizeof(...)
            getsizeof(object, default) -> int
            
            Return the size of object in bytes.
        
        gettrace(...)
            gettrace()
            
            Return the global debug tracing function set with sys.settrace.
            See the debugger chapter in the library manual.
        
        getwindowsversion(...)
            getwindowsversion()
            
            Return information about the running version of Windows as a named tu
ple.
            The members are named: major, minor, build, platform, service_pack,
            service_pack_major, service_pack_minor, suite_mask, and product_type.
 For
            backward compatibility, only the first 5 items are available by index
ing.
            All elements are numbers, except service_pack which is a string. Plat
form
            may be 0 for win32s, 1 for Windows 9x/ME, 2 for Windows NT/2000/XP/Vi
```

```
sta/7,
        3 for Windows CE. Product_type may be 1 for a workstation, 2 for a do
main
        controller, 3 for a server.

    setcheckinterval(...)
        setcheckinterval(n)

        Tell the Python interpreter to check for asynchronous events every
        n instructions.  This also affects how often thread switches occur.

    setprofile(...)
        setprofile(function)

        Set the profiling function.  It will be called on each function call
        and return.  See the profiler chapter in the library manual.

    setrecursionlimit(...)
        setrecursionlimit(n)

        Set the maximum depth of the Python interpreter stack to n.  This
        limit prevents infinite recursion from causing an overflow of the C
        stack and crashing Python.  The highest possible limit is platform-
        dependent.

    settrace(...)
        settrace(function)

        Set the global debug tracing function.  It will be called on each
        function call.  See the debugger chapter in the library manual.

DATA
    __stderr__ = <open file '<stderr>', mode 'w'>
    __stdin__ = <open file '<stdin>', mode 'r'>
    __stdout__ = <open file '<stdout>', mode 'w'>
    api_version = 1013
    argv = [r'c:\python27\lib\site-packages\ipykernel\__main__.py', '-f', ...
    builtin_module_names = ('__builtin__', '__main__', '_ast', '_bisect', ...
    byteorder = 'little'
    copyright = 'Copyright (c) 2001-2016 Python Software Foundati...ematis...
    displayhook = <ipykernel.displayhook.ZMQShellDisplayHook object>
    dllhandle = 503054336
    dont_write_bytecode = False
    exc_value = TypeError("<module 'sys' (built-in)> is a built-in module"...
    exec_prefix = r'c:\python27'
    executable = r'c:\python27\python.exe'
    flags = sys.flags(debug=0, py3k_warning=0, division_warn...unicode=0, ...
    float_info = sys.float_info(max=1.7976931348623157e+308, max_...epsilo...
    float_repr_style = 'short'
    hexversion = 34016496
    last_value = TypeError('<lambda>() takes exactly 2 arguments (1 given)...
    long_info = sys.long_info(bits_per_digit=15, sizeof_digit=2)
    maxint = 2147483647
    maxsize = 2147483647
    maxunicode = 65535
    meta_path = [<six._SixMetaPathImporter object>, <pkg_resources.extern....
    modules = {'IPython': <module 'IPython' from 'c:\python27\lib\site-pac...
```

```
path = ['', r'C:\Windows\system32\python27.zip', r'c:\python27\DLLs', ...
path_hooks = [<type 'zipimport.zipimporter'>]
path_importer_cache = {'': None, r'C:\Users\PRAUD01\.ipython': None, r...
platform = 'win32'
prefix = r'c:\python27'
ps1 = 'In : '
ps2 = '...: '
ps3 = 'Out: '
py3kwarning = False
stderr = <ipykernel.iostream.OutStream object>
stdin = <open file '<stdin>', mode 'r'>
stdout = <ipykernel.iostream.OutStream object>
subversion = ('CPython', '', '')
version = '2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v...
version_info = sys.version_info(major=2, minor=7, micro=12, releaselev...
warnoptions = []
winver = '2.7'
```

Methods of importing

```
import sys
from sys import *           # Not recommended by PEP 8
from sys import version
from sys import version, getsizeof
from sys import version as vr    # alias importing
```

In [1112]:
```
import sys
sys.version
```

Out[1112]:
```
'2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Int
el)]'
```

In [1113]:
```
del sys    # 'sys' object will be removed from the heap memory
```

In [1114]:
```
sys
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1114-d7099a5e9c44> in <module>()
----> 1 sys

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

In [1115]:
```
from sys import version
print "version = ", version
```

```
version =  2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 3
2 bit (Intel)]
```

```
In [1116]: from sys import version as vr

           print "version is ", vr

           version is  2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500
            32 bit (Intel)]
```

```
In [1117]: del vr     # removes 'vr' object from the heap memory
```

```
In [1118]: vr

           ---------------------------------------------------------------------------
           NameError                                 Traceback (most recent call last)
           <ipython-input-1118-8264846d88d6> in <module>()
           ----> 1 vr

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

Note: Selective imports optimizes the memory usage; but care should be taken when user-defined identifiers in the script/project have the same names as these functions of modules

```
In [1119]: import os
```

```
In [1120]: os.system("echo 'Hello World!'")    # 0 means the statement executed properly.
            Output will be displayed in console; not here

Out[1120]: 0
```

```
In [1121]: combinedDir = os.path.join('first', 'second', 'third', 'fourth')
           print 'combinedDir is ', combinedDir

           combinedDir is  first\second\third\fourth
```

```
In [1122]: print os.path.exists(combinedDir)

           False
```

```
  In [91]: os.getcwd()

  Out[91]: 'C:\\Users\\HOME\\Google Drive\\python\\tut\\complete Material'
```

```
In [1124]: os.mkdir('newFolder')
```

```
  In [93]: os.listdir(os.getcwd())

  Out[93]: ['.ipynb_checkpoints',
            'desktop.ini',
            'newFolder',
            'python - Basic Level Module.ipynb',
            'Untitled.ipynb']
```

```
In [1125]: os.makedirs(combinedDir)   # Common gotcha: It is 'makedirs' and not 'mkdirs'
In [95]: os.listdir(os.getcwd())
```

```
Out[95]: ['.ipynb_checkpoints',
          'desktop.ini',
          'first',
          'newFolder',
          'python - Basic Level Module.ipynb',
          'Untitled.ipynb']
```

```
In [1126]: with open('myTest.txt', 'ab+') as myTst:
               myTst.write("Will power can do a lot of things!")
               myTst.close()


           # working with files will be done in next chapter.
```

```
In [97]: os.listdir(os.getcwd())
```

```
Out[97]: ['.ipynb_checkpoints',
          'desktop.ini',
          'first',
          'myTest.txt',
          'newFolder',
          'python - Basic Level Module.ipynb',
          'Untitled.ipynb']
```

```
In [1127]: os.rename('myTest.txt', 'myNewFile.tsf')   # renaming files
```

```
In [99]: os.listdir(os.getcwd())
```

```
Out[99]: ['.ipynb_checkpoints',
          'desktop.ini',
          'first',
          'myNewFile.tsf',
          'newFolder',
          'python - Basic Level Module.ipynb',
          'Untitled.ipynb']
```

```
In [1128]: os.rename('newFolder', 'myNewFolder')    # renaming directories
```

```
In [101]: os.listdir(os.getcwd())
```

```
Out[101]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myNewFile.tsf',
           'myNewFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [1129]: os.chdir('first/')
In [1130]: os.listdir(os.getcwd())

Out[1130]: ['desktop.ini', 'second']
```

```
In [1131]: os.chdir('..')    # changing to previous directories
```

```
 In [105]: os.getcwd()

 Out[105]: 'C:\\Users\\HOME\\Google Drive\\python\\tut\\complete Material'
```

```
 In [106]: os.listdir(os.getcwd())

 Out[106]: ['.ipynb_checkpoints',
            'desktop.ini',
            'first',
            'myNewFile.tsf',
            'myNewFolder',
            'python - Basic Level Module.ipynb',
            'Untitled.ipynb']
```

```
In [1132]: os.stat('myNewFile.tsf')   # for more details, goto help(os.stat('myNewFile.ts
           f'))

Out[1132]: nt.stat_result(st_mode=33206, st_ino=0L, st_dev=0L, st_nlink=0, st_uid=0, st_
           gid=0, st_size=34L, st_atime=1482670774L, st_mtime=1482670774L, st_ctime=1482
           079578L)
```

```
In [1133]: modifiedTime = os.stat('myNewFile.tsf').st_mtime

           print "myNewFile.tsf was last modified on ", modifiedTime  # epoch time

           myNewFile.tsf was last modified on  1482670774.15
```

```
In [1134]: from datetime import datetime
           print datetime.fromtimestamp(modifiedTime)

           2016-12-25 18:29:34.148554
```

```
In [1135]: print "myNewFile.tsf was last modified on ", datetime.fromtimestamp(modifiedTi
           me)

           myNewFile.tsf was last modified on  2016-12-25 18:29:34.148554
```

In [1136]:
```
#!/usr/bin/python

import sys
import os

if sys.platform == 'win32':
    dirToCheck = raw_input("Enter a directory path to check :") #'C:\Python27
\Tools'
else:
    dirToCheck = raw_input("Enter a directory path to check :")

for dirpath, dirnames, filenames in os.walk(dirToCheck):
        print 'Current Path:', dirpath
        print 'Directories:', dirnames
        print 'Files:', filenames
        print '-'*50
```

```
Enter a directory path to check :C:\Python27\Tools
Current Path: C:\Python27\Tools
Directories: ['i18n', 'pynche', 'Scripts', 'versioncheck', 'webchecker']
Files: []
--------------------------------------------------
Current Path: C:\Python27\Tools\i18n
Directories: []
Files: ['makelocalealias.py', 'msgfmt.py', 'pygettext.py']
--------------------------------------------------
Current Path: C:\Python27\Tools\pynche
Directories: ['X']
Files: ['ChipViewer.py', 'ColorDB.py', 'DetailsViewer.py', 'html40colors.tx
t', 'ListViewer.py', 'Main.py', 'namedcolors.txt', 'pyColorChooser.py', 'pync
he.pyw', 'PyncheWidget.py', 'README.txt', 'StripViewer.py', 'Switchboard.py',
 'TextViewer.py', 'TypeinViewer.py', 'webcolors.txt', 'websafe.txt', '__init_
_.py']
--------------------------------------------------
Current Path: C:\Python27\Tools\pynche\X
Directories: []
Files: ['rgb.txt', 'xlicense.txt']
--------------------------------------------------
Current Path: C:\Python27\Tools\Scripts
Directories: []
Files: ['2to3.py', 'analyze_dxp.py', 'byext.py', 'byteyears.py', 'checkappen
d.py', 'checkpip.py', 'checkpyc.py', 'classfix.py', 'cleanfuture.py', 'combin
erefs.py', 'copytime.py', 'crlf.py', 'cvsfiles.py', 'db2pickle.py', 'diff.p
y', 'dutree.py', 'eptags.py', 'finddiv.py', 'findlinksto.py', 'findnocoding.p
y', 'find_recursionlimit.py', 'fixcid.py', 'fixdiv.py', 'fixheader.py', 'fixn
otice.py', 'fixps.py', 'google.py', 'gprof2html.py', 'h2py.py', 'hotshotmain.
py', 'ifdef.py', 'lfcr.py', 'linktree.py', 'lll.py', 'logmerge.py', 'mailerda
emon.py', 'md5sum.py', 'methfix.py', 'mkreal.py', 'ndiff.py', 'nm2def.py', 'o
bjgraph.py', 'parseentities.py', 'patchcheck.py', 'pathfix.py', 'pdeps.py',
 'pickle2db.py', 'pindent.py', 'ptags.py', 'pydocgui.pyw', 'pysource.py', 'RE
ADME.txt', 'redemo.py', 'reindent-rst.py', 'reindent.py', 'rgrep.py', 'serve.
py', 'setup.py', 'suff.py', 'svneol.py', 'texcheck.py', 'texi2html.py', 'tree
sync.py', 'untabify.py', 'which.py', 'win_add2path.py', 'xxci.py']
--------------------------------------------------
Current Path: C:\Python27\Tools\versioncheck
Directories: []
Files: ['checkversions.py', 'pyversioncheck.py', 'README.txt', '_checkversio
n.py']
--------------------------------------------------
Current Path: C:\Python27\Tools\webchecker
Directories: []
Files: ['README.txt', 'tktools.py', 'wcgui.py', 'wcmac.py', 'webchecker.py',
 'websucker.py', 'wsgui.py']
--------------------------------------------------
```

Assignment : exceute this, and observe the output

`

```
for i in os.environ:  # To get the environmental variables
    print i
```

```
In [1137]: [en for en in os.environ]
```

```
Out[1137]:
```

```
['TMP',
 'COMPUTERNAME',
 'USERDOMAIN',
 'GOROOT',
 'DEFLOGDIR',
 'PSMODULEPATH',
 'CAI_MSQ',
 'COMMONPROGRAMFILES',
 'PROCESSOR_IDENTIFIER',
 'VBOX_MSI_INSTALL_PATH',
 'PROGRAMFILES',
 'PROCESSOR_REVISION',
 'PATH',
 'SYSTEMROOT',
 'CLICOLOR',
 'PROGRAMFILES(X86)',
 'SDROOT',
 'MPLBACKEND',
 'TERM',
 'TEMP',
 'COMMONPROGRAMFILES(X86)',
 'PROCESSOR_ARCHITECTURE',
 'LEGALCOUNTRYID',
 'ALLUSERSPROFILE',
 'LOCALAPPDATA',
 'HOMEPATH',
 'JPY_INTERRUPT_EVENT',
 'PROGRAMW6432',
 'USERNAME',
 'LOGONSERVER',
 'PROMPT',
 'WINDOWS_TRACING_FLAGS',
 'JPY_PARENT_PID',
 'PROGRAMDATA',
 'TOUCHAPPSTARGETDIR',
 'VSEDEFLOGDIR',
 'USERDNSDOMAIN',
 'GIT_PAGER',
 'SESSIONNAME',
 'PATHEXT',
 'FP_NO_HOST_CHECK',
 'WINDIR',
 'CSAM_SOCKADAPTER',
 'WINDOWS_TRACING_LOGFILE',
 'HOMEDRIVE',
 'PAGER',
 'SYSTEMDRIVE',
 'CSAM_LOGGER_CONF',
 'COMSPEC',
 'NUMBER_OF_PROCESSORS',
 'APPDATA',
 'CAI_CAFT',
 'PROCESSOR_LEVEL',
 'PROCESSOR_ARCHITEW6432',
 'COMMONPROGRAMW6432',
 'OS',
 'PUBLIC',
 'IPY_INTERRUPT_EVENT',
 'USERPROFILE']
```

In [118]:
```
print "os.environ.get('TMP') is ", os.environ.get('TMP')
```
```
os.environ.get('TMP') is  C:\Users\HOME\AppData\Local\Temp
```

In [3]:
```
import os
print "os.environ.get('USERPROFILE') is ", os.environ.get('USERPROFILE')
```
```
 os.environ.get('USERPROFILE') is  C:\Users\HOME
```

In [4]:
```
print "os.environ.get('USERNAME') is ", os.environ.get('USERNAME')
```
```
os.environ.get('USERNAME') is  HOME
```

In [121]:
```
print "os.environ.get('TEMP') is ", os.environ.get('TEMP')
```
```
os.environ.get('TEMP') is  C:\Users\HOME\AppData\Local\Temp
```

In [122]:
```
enVar = os.environ
print type(enVar)
```
```
<type 'instance'>
```

In [5]:
```
os
```
Out[5]:
```
<module 'os' from 'c:\python27\lib\os.pyc'>
```

In [9]:
```
filePath = os.path.join(os.environ.get('TMP'), 'test.txt')
print filePath
```
```
C:\Users\HOME\AppData\Local\Temp\test.txt
```

In [124]:
```
print os.path.exists(filePath)
```
```
False
```

In [125]:
```
with open(filePath, 'ab+') as f:
    f.write("This is the right time to invest my energies, to get success")
    f.close()
```

In [126]:
```
print os.path.exists(filePath)
```
```
True
```

In [127]:
```
os.path.basename(filePath)
```
Out[127]:
```
'test.txt'
```

In [128]:
```
os.path.dirname(filePath)
```
Out[128]:
```
'C:\\Users\\HOME\\AppData\\Local\\Temp'
```

```
In [10]: os.path.dirname(filePath) + os.path.sep + os.path.basename(filePath)

Out[10]: 'C:\\Users\\HOME\\AppData\\Local\\Temp\\test.txt'
```

```
In [130]: filePath

Out[130]: 'C:\\Users\\HOME\\AppData\\Local\\Temp\\test.txt'
```

```
In [131]: os.path.splitext(filePath)

Out[131]: ('C:\\Users\\HOME\\AppData\\Local\\Temp\\test', '.txt')
```

```
In [132]: os.path.splitext('someRandomfile.pdf')

Out[132]: ('someRandomfile', '.pdf')
```

```
In [133]: os.listdir(os.getcwd())

Out[133]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myNewFile.tsf',
           'myNewFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [134]: os.listdir('.')

Out[134]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myNewFile.tsf',
           'myNewFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [135]: os.mkdir('newFolder')
```

```
In [136]: os.listdir('.')

Out[136]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [137]: os.chdir('newFolder/')
          os.listdir('..')

Out[137]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [138]: os.getcwd()

Out[138]: 'C:\\Users\\HOME\\Google Drive\\python\\tut\\complete Material\\newFolder'
```

time related modules - time, datetime, pytz, ...

```
In [1138]: import time
```

```
In [1139]: print dir(time)

['__doc__', '__name__', '__package__', 'accept2dyear', 'altzone', 'asctime',
 'clock', 'ctime', 'daylight', 'gmtime', 'localtime', 'mktime', 'sleep', 'str
ftime', 'strptime', 'struct_time', 'time', 'timezone', 'tzname']
```

In [1140]:
```
print time.__doc__
```

```
This module provides various functions to manipulate time values.

There are two standard representations of time.  One is the number
of seconds since the Epoch, in UTC (a.k.a. GMT).  It may be an integer
or a floating point number (to represent fractions of seconds).
The Epoch is system-defined; on Unix, it is generally January 1st, 1970.
The actual value can be retrieved by calling gmtime(0).

The other representation is a tuple of 9 integers giving local time.
The tuple items are:
  year (four digits, e.g. 1998)
  month (1-12)
  day (1-31)
  hours (0-23)
  minutes (0-59)
  seconds (0-59)
  weekday (0-6, Monday is 0)
  Julian day (day in the year, 1-366)
  DST (Daylight Savings Time) flag (-1, 0 or 1)
If the DST flag is 0, the time is given in the regular time zone;
if it is 1, the time is given in the DST time zone;
if it is -1, mktime() should guess based on the date and time.

Variables:

timezone -- difference in seconds between UTC and local standard time
altzone -- difference in  seconds between UTC and local DST time
daylight -- whether local time should reflect DST
tzname -- tuple of (standard time zone name, DST time zone name)

Functions:

time() -- return current time in seconds since the Epoch as a float
clock() -- return CPU time since process start as a float
sleep() -- delay for a number of seconds given as a float
gmtime() -- convert seconds since Epoch to UTC tuple
localtime() -- convert seconds since Epoch to local time tuple
asctime() -- convert time tuple to string
ctime() -- convert time in seconds to string
mktime() -- convert local time tuple to seconds since Epoch
strftime() -- convert time tuple to string according to format specification
strptime() -- parse string to time tuple according to format specification
tzset() -- change the local timezone
```

In [1141]:
```
time.tzname
```

Out[1141]:
```
('India Standard Time', 'India Daylight Time')
```

In [1142]:
```
time.tzname[0]
```

Out[1142]:
```
'India Standard Time'
```

```
In [1143]: time.daylight # Results in boolean result of existence or absence of DST, in t
           hat time zone

Out[1143]: 0

In [1144]: time.timezone

Out[1144]: -19800

In [1145]: time.time()    #seconds past from epoch time, till now

Out[1145]: 1482684674.557

In [1146]: time.ctime()

Out[1146]: 'Sun Dec 25 22:22:10 2016'

In [1147]: time.asctime()

Out[1147]: 'Sun Dec 25 22:22:27 2016'

In [1148]: type(time.asctime())

Out[1148]: str

In [1149]: time.gmtime()

Out[1149]: time.struct_time(tm_year=2016, tm_mon=12, tm_mday=25, tm_hour=16, tm_min=54,
            tm_sec=24, tm_wday=6, tm_yday=360, tm_isdst=0)

In [1150]: type(time.gmtime())

Out[1150]: time.struct_time

In [1151]: time.localtime()

Out[1151]: time.struct_time(tm_year=2016, tm_mon=12, tm_mday=25, tm_hour=23, tm_min=1, t
           m_sec=55, tm_wday=6, tm_yday=360, tm_isdst=0)

In [1152]: t = time.localtime()
           print type(t)

           <type 'time.struct_time'>

In [1153]: time.clock()

Out[1153]: 1.5205234554047576e-06
```

Assignment : what is the difference between time.time() and time.clock()

```
In [1154]: time.sleep(6)  # To let the interpreter to sleep for 6 seconds
In [1155]: time.strptime('Fri Aug 19 07:33:01 2016')   # String to time tuple conversion

Out[1155]: time.struct_time(tm_year=2016, tm_mon=8, tm_mday=19, tm_hour=7, tm_min=33, tm
           _sec=1, tm_wday=4, tm_yday=232, tm_isdst=-1)
```

```
In [1156]: time.strptime(time.asctime())

Out[1156]: time.struct_time(tm_year=2016, tm_mon=12, tm_mday=25, tm_hour=23, tm_min=2, t
           m_sec=24, tm_wday=6, tm_yday=360, tm_isdst=-1)
```

```
In [1157]: time.strptime(time.ctime())

Out[1157]: time.struct_time(tm_year=2016, tm_mon=12, tm_mday=25, tm_hour=23, tm_min=2, t
           m_sec=27, tm_wday=6, tm_yday=360, tm_isdst=-1)
```

Assignment : what is the difference between time.asctime() and time.ctime()?

```
In [1158]: time.strptime("8/4/1988", "%d/%m/%Y")

Out[1158]: time.struct_time(tm_year=1988, tm_mon=4, tm_mday=8, tm_hour=0, tm_min=0, tm_s
           ec=0, tm_wday=4, tm_yday=99, tm_isdst=-1)
```

```
In [1159]: time.strptime("08 Apr 1988", "%d %b %Y")

Out[1159]: time.struct_time(tm_year=1988, tm_mon=4, tm_mday=8, tm_hour=0, tm_min=0, tm_s
           ec=0, tm_wday=4, tm_yday=99, tm_isdst=-1)
```

```
In [1160]: time.strptime("15-Aug-1947", "%d-%b-%Y")

Out[1160]: time.struct_time(tm_year=1947, tm_mon=8, tm_mday=15, tm_hour=0, tm_min=0, tm_
           sec=0, tm_wday=4, tm_yday=227, tm_isdst=-1)
```

```
In [1161]: myTime = time.ctime()
           print myTime
           newCreatedTime = time.mktime(time.strptime(myTime))
           print newCreatedTime

           Sun Dec 25 23:02:44 2016
           1482687164.0
```

Interesting Article: https://wiki.python.org/moin/WorkingWithTime (https://wiki.python.org/moin/WorkingWithTime)

```
In [1162]: import datetime
```

```
In [1163]: print datetime.__doc__

           Fast implementation of the datetime type.
```

```
In [1164]: print dir(datetime)
```

```
['MAXYEAR', 'MINYEAR', '__doc__', '__name__', '__package__', 'date', 'datetim
e', 'datetime_CAPI', 'time', 'timedelta', 'tzinfo']
```

```
In [1165]: print datetime.datetime.__doc__
```

```
datetime(year, month, day[, hour[, minute[, second[, microsecond[,tzinf
o]]]]])

The year, month and day arguments are required. tzinfo may be None, or an
instance of a tzinfo subclass. The remaining arguments may be ints or longs.
```

```
In [1166]: print dir(datetime.datetime)
```

```
['__add__', '__class__', '__delattr__', '__doc__', '__eq__', '__format__', '_
_ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt
__', '__ne__', '__new__', '__radd__', '__reduce__', '__reduce_ex__', '__repr_
_', '__rsub__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclas
shook__', 'astimezone', 'combine', 'ctime', 'date', 'day', 'dst', 'fromordina
l', 'fromtimestamp', 'hour', 'isocalendar', 'isoformat', 'isoweekday', 'max',
 'microsecond', 'min', 'minute', 'month', 'now', 'replace', 'resolution', 'se
cond', 'strftime', 'strptime', 'time', 'timetuple', 'timetz', 'today', 'toord
inal', 'tzinfo', 'tzname', 'utcfromtimestamp', 'utcnow', 'utcoffset', 'utctim
etuple', 'weekday', 'year']
```

```
In [1167]: datetime.datetime.now()   # Local time

Out[1167]: datetime.datetime(2016, 12, 25, 23, 3, 2, 896000)
```

```
In [1168]: type(datetime.datetime.now())

Out[1168]: datetime.datetime
```

```
In [1169]: datetime.datetime.utcnow()

Out[1169]: datetime.datetime(2016, 12, 25, 17, 33, 9, 506000)
```

```
In [1170]: datetime.date.today()

Out[1170]: datetime.date(2016, 12, 25)
```

```
In [1171]: tdy = datetime.date.today()
           print tdy
```

```
2016-12-25
```

```
In [1172]: print tdy.strftime("four-digit year: %Y, two-digit year: %y, month: %m, day: %
           d, seconds: %S")
```

```
four-digit year: 2016, two-digit year: 16, month: 12, day: 25, seconds: 00
```

```
In [1173]: print tdy.strftime("four-digit year: %Y, two-digit year: %y, month: %m, monthI
            nWords: %b, day: %d")
```

```
four-digit year: 2016, two-digit year: 16, month: 12, monthInWords: Dec, day:
 25
```

NOTE: Both time and datetime modules can be used together

```
In [1174]: t = datetime.datetime.now() # For the local timezone
           print t

           print t.timetuple()

           print "Epoch Seconds:", time.mktime(t.timetuple())
```

```
2016-12-25 23:03:34.568000
time.struct_time(tm_year=2016, tm_mon=12, tm_mday=25, tm_hour=23, tm_min=3, t
m_sec=34, tm_wday=6, tm_yday=360, tm_isdst=-1)
Epoch Seconds: 1482687214.0
```

```
In [1175]: t = datetime.datetime.utcnow()   # For UTC
           print t
           print "Epoch Seconds:", time.mktime(t.timetuple())
```

```
2016-12-25 17:33:38.483000
Epoch Seconds: 1482667418.0
```

timeit module

```
In [1176]: import timeit
```

```
In [1177]: logic = '[x for x in xrange(10) if x%2 != 0]'

           eval(logic)   # eval() - builtin function to execute a statement
```

```
Out[1177]: [1, 3, 5, 7, 9]
```

```
In [1178]: t = timeit.Timer(logic)
           print t
           print "1000 repeats of this logic takes :", t.timeit(1000), " seconds"
```

```
<timeit.Timer instance at 0x03990DC8>
1000 repeats of this logic takes : 0.00187518555137  seconds
```

```
In [1179]: print "10,00,000 repeats of this logic takes :", t.timeit(1000000), " seconds"
```

```
10,00,000 repeats of this logic takes : 2.08296166038  seconds
```

```
In [1180]: timeit.timeit('range(12)')
```

```
Out[1180]: 0.5930763724719839
```

```
In [1181]: timeit.timeit('xrange(12)')
```

```
Out[1181]: 0.3552208883430126
```

Python file types

.pyw - This is windows executable

.pyc - compiled python bytecode file, for a particular .py file.

.pyd - python dll file

.pyc file is platform-independent, yet interpreter dependent. The interpreter checks for the *.py* file last modified time stamp with that of *.pyc* file. If there is a mismatch, then that *.pyc* file will be discarded, and a new *.pyc* file will be created.

It is created either

```
1. when a particular _.py_ file is imported in another python script and/or in pyth
on interpreter.
2. Manually _.pyc_ file can be created, when the _.py_ file is compiled using py_co
mpile
    python -m py_compile fileName.py
```

```
In [1182]: import os
```

```
In [187]: os.listdir('.')
```

```
Out[187]: ['.ipynb_checkpoints',
 'desktop.ini',
 'first',
 'myNewFile.tsf',
 'myNewFolder',
 'newFolder',
 'python - Basic Level Module.ipynb',
 'Untitled.ipynb']
```

```
In [1183]: code = "\
with open('myFile.txt', 'ab+') as myf:\
    myf.write('This is my final struggle to succes. So, I will give my best')\
    myf.close()\
"



with open('myfile.pyw', 'ab+') as f:
    f.write(code)
```

```
In [189]: os.listdir('.')

Out[189]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myfile.pyw',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

INFERENCE: .pyw is executable in windows, and it can be used when no i/o operations are involved for the script

In [190]:
```
#!/usr/bin/python

'''
        Purpose: module importing demonstration

'''
# newScript.py

vowels = 'aeiou'

luckyNumber = 1321

def firstFunction():
        '''
        This is firstFunction
        :return: None
        '''
        print "This is first function"


def addition(a,b):
        '''
                performs addition operation
                ex: addition(12, 34)
                returns: a+b
        '''
        return a+b

def subtraction(a,b):
        '''
                performs subtraction operation
        '''
        return a-b

def multiplication(a,b):
        '''
                performs multiplication operation
        '''
        return a*b


if __name__ == '__main__':
        print 'This script is executed directly'
else:
    print 'This script is imported from another module'
```

```
This script is executed directly
```

```
In [193]: os.listdir('.')

Out[193]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myfile.pyw',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'newScript.py',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [194]: import newScript     #.py extension should not be given

          This script is imported from another module
```

```
In [195]: os.listdir('.')      # observe the .pyc file

Out[195]: ['.ipynb_checkpoints',
           'desktop.ini',
           'first',
           'myfile.pyw',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'newScript.py',
           'newScript.pyc',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']
```

```
In [196]: print newScript.__doc__

                  Purpose: module importing demonstration
```

```
In [197]: print dir(newScript)

          ['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'additio
          n', 'firstFunction', 'luckyNumber', 'multiplication', 'subtraction', 'vowel
          s']
```

```
In [198]: newScript.firstFunction()

          This is first function
```

```
In [199]: help(newScript.addition)

          Help on function addition in module newScript:

          addition(a, b)
              performs addition operation
              ex: addition(12, 34)
              returns: a+b
```

```
In [200]: newScript.addition(99, 67)

Out[200]: 166
```

callable() buitlin function; results whether a particular object is callable or not

```
In [201]: callable(newScript.addition)

Out[201]: True
```

```
In [202]: newScript.addition(12.23, 6.07)

Out[202]: 18.3
```

```
In [203]: newScript.subtraction(12.23, 6.07)

Out[203]: 6.16
```

The newScript.py file was modified with better help for functions. Also, two static variables were added.

```
In [204]: help(newScript.addition)

          Help on function addition in module newScript:

          addition(a, b)
              performs addition operation
              ex: addition(12, 34)
              returns: a+b
```

Notice that the changes were not reflected.

Modifying the script of load modules needs reloading the module to get the changes to be affected in the working script, or interpreter.

```
In python 2.x,
    reload(<user-defined Module name>)
    ex: reload(newScript)
In python 3.x,
    import imp
    imp.reload(<user-defined Module name>)
    imp.reload(newScript)

    or

    import importlib
    importlib.reload(<user-defined Module name>)
    importlib.reload(newScript)
```

There are various other modules like xreload, reimport with additional functionalities, for reloading the modules.

```
In [205]: reload(newScript)   # To get the changes
```

```
This script is imported from another module
```

```
Out[205]: <module 'newScript' from 'newScript.pyc'>
```

```
In [206]: help(newScript.addition)
```

```
Help on function addition in module newScript:

addition(a, b)
    performs addition operation
    ex: addition(12, 34)
    returns: a+b
```

```
In [207]: print dir(newScript)
```

```
['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'additio
n', 'firstFunction', 'luckyNumber', 'multiplication', 'subtraction', 'vowel
s']
```

```
In [208]: newScript.luckyNumber
```

```
Out[208]: 1321
```

To ensure that certain part of logic should be executed only when the script is independently called, write that logic under `if __name__ == '__main__'` condition

```
In [209]: reload(newScript)

          This script is imported from another module

Out[209]: <module 'newScript' from 'newScript.pyc'>
```

At times, if the imported module has any dependencies on other imported modules, those functionality will not get refreshed.

In that case, it would be better to delete that imported module object.

```
#del <importedModuleName>
del newScript

or
# using sys module
import sys
#del sys.modules[<importedModuleName>]
del sys.modules[newScript]
```

```
In [210]: del newScript   # deletes the imported object from the interpreter cache

In [211]: newScript       # results in error, as that object is no more present in inter
          preter cache

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-211-9b0d1d72cb8a> in <module>()
          ----> 1 newScript       # results in error, as that object is no more present
           in interpreter cache

          NameError: name 'newScript' is not defined

In [212]: import newScript
          print dir(newScript)

          ['__builtins__', '__doc__', '__file__', '__name__', '__package__', 'additio
          n', 'firstFunction', 'luckyNumber', 'multiplication', 'subtraction', 'vowel
          s']

In [213]: newScript.__file__   # Observe that the compiled bytecode object is called; no
          t the .py file

Out[213]: 'newScript.pyc'

In [214]: newScript.__doc__

Out[214]: '\n\tPurpose: module importing demonstration\n\t\n'
```

```
In [215]: print newScript.__doc__
```

```
        Purpose: module importing demonstration
```

```
In [216]: help(newScript)
```

```
Help on module newScript:

NAME
    newScript - Purpose: module importing demonstration

FILE
    c:\users\home\google drive\python\tut\complete material\newscript.py

FUNCTIONS
    addition(a, b)
        performs addition operation
        ex: addition(12, 34)
        returns: a+b

    firstFunction()
        This is firstFunction
        :return: None

    multiplication(a, b)
        performs multiplication operation

    subtraction(a, b)
        performs subtraction operation

DATA
    luckyNumber = 1321
    vowels = 'aeiou'
```

NOTE: Ensure that the script related docstrings were written, just after the shebang line.

eval,exec, execfile, compile and py_compile

eval(str,globals,locals) - This function executes an expression string and returns the result.

```
In [1]: eval('2+3')   # returns the output
```

```
Out[1]: 5
```

```
In [2]: eval("'Python'*3") # observe both single(') and double(") quotes
```

```
Out[2]: 'PythonPythonPython'
```

```
In [3]: eval("eval('2+3')")

Out[3]: 5
```

```
In [4]: eval("print 'Hello World!'")

          File "<string>", line 1
            print 'Hello World!'
                ^
        SyntaxError: invalid syntax
```

Inference: statement execution is not possible with eval()

exec(filename,globals,locals) - function executes the statements

```
In [5]: exec("print 'Hello World!'")  # exec() executes a string containing arbitrary
         python code

        Hello World!
```

```
In [6]: exec('2+34')  # doesn't return output
```

```
In [7]: exec("exec('2+34')")  # doesn't return output
```

Interview Question : what is the difference between eval() and exec() ?

```
In [8]: a=[1,2,3,45]

        exec "for i in a: print i"

        1
        2
        3
        45
```

```
In [9]: exec '\
        a=[1,2,3,45];\
        exec "for i in a: print i"'

        1
        2
        3
        45
```

```
In [10]: exec("'Hello World!'")
```

```
In [11]: exec("print 'Hello World!'")

         Hello World!
```

execfile(filename,globals,locals) - function executes the contents of a file

```
In [12]: #!/usr/bin/python
         # fileName.py
         print "HelloWorld!"

         birds=['parrot','hen','hyna']
         for b in birds:
                 print b
```

```
HelloWorld!
parrot
hen
hyna
```

```
In [13]: del birds, b
```

```
In [14]: import os;
         os.listdir(os.getcwd())
```

```
Out[14]: ['.ipynb_checkpoints',
          'desktop.ini',
          'fibGenerator.py',
          'fibGenerator.pyc',
          'fileName.py',
          'first',
          'myfile.pyw',
          'myNewFile.tsf',
          'myNewFolder',
          'newFolder',
          'newScript.py',
          'newScript.pyc',
          'python - Basic Level Module.ipynb',
          'python+-+Basic+Level+Module.html']
```

Observe that 'fileName.py' is placed in the current working directory

```
In [15]: execfile("fileName.py")  # To execute the python script
```

```
HelloWorld!
parrot
hen
hyna
```

```
In [16]: myGlobals={'x':7,'y':10,'birds':['parrot','pigeon','sparrow']}

         myLocals={'x':2,'y':20}
```

In [17]:
```
a=eval("3*x+4*y",myGlobals,myLocals)     # locals are preferred - 3*2+ 4*20  =
 86

print a
```
```
86
```

In [18]:
```
a=eval("3*x+4*y",myLocals, myGlobals)
                     # here, also, locals are preferred.
                     # syntax: eval(string, global_variables, local_variables)

print a
```
```
61
```

In [19]:
```
myLocals ={}

a=eval("3*x+4*y",myLocals, myGlobals)   # In the absence of locals, globals ar
e choosen

print a
```
```
61
```

In [20]:
```
myLocals ={'birds':['localBird','myLocal Bird','sparrow']}
```

In [21]:
```
exec "for b in birds: print b" in myGlobals,myLocals
```
```
localBird
myLocal Bird
sparrow
```

In [22]:
```
exec "for b in birds: print b" in myLocals, myGlobals # preference will be don
e based on position
```
```
parrot
pigeon
sparrow
```

In [23]:
```
execfile("fileName.py",myGlobals,myLocals) # Values present within script are
 preferred. there are more local
```
```
HelloWorld!
parrot
hen
hyna
```

Observe that the values for 'birds' is preferred to from 'fileName.py'

In [24]:
```
del birds
```

In [25]:
```
#!/usr/bin/python
# fileName.py
print "HelloWorld!"


for b in birds:
        print b
```

```
HelloWorld!

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-25-74f39257017d> in <module>()
      4
      5
----> 6 for b in birds:
      7         print b

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

In [26]:
```
execfile("fileName.py",myGlobals,myLocals)  # problem with ipython environmen
t, as filename.py is still not updated
```

```
HelloWorld!
parrot
hen
hyna
```

In [30]:
```
del fileName
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-30-574981b7b5bd> in <module>()
----> 1 del fileName

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

In [31]:
```
execfile("fileName.py",myGlobals,myLocals)  # problem with ipython environmen
t, as filename.py is still not updated
```

```
HelloWorld!
parrot
hen
hyna
```

In [32]:
```
execfile("fileName.py",myGlobals,myLocals)  # problem with ipython environmen
t, as filename.py is still not updated
```

```
HelloWorld!
parrot
hen
hyna
```

In [33]:
```
myLocals = {'birds':['ostrich','vulchur','Bat']}
```

```
In [34]: execfile("fileName.py",myGlobals,myLocals)

         HelloWorld!
         ostrich
         vulchur
         Bat

In [35]: myLocals

Out[35]: {'b': 'Bat', 'birds': ['ostrich', 'vulchur', 'Bat']}
```

```
In [36]: myGlobals
```

```
Out[36]:
```

```
{'__builtins__': {'ArithmeticError': ArithmeticError,
  'AssertionError': AssertionError,
  'AttributeError': AttributeError,
  'BaseException': BaseException,
  'BufferError': BufferError,
  'BytesWarning': BytesWarning,
  'DeprecationWarning': DeprecationWarning,
  'EOFError': EOFError,
  'Ellipsis': Ellipsis,
  'EnvironmentError': EnvironmentError,
  'Exception': Exception,
  'False': False,
  'FloatingPointError': FloatingPointError,
  'FutureWarning': FutureWarning,
  'GeneratorExit': GeneratorExit,
  'IOError': IOError,
  'ImportError': ImportError,
  'ImportWarning': ImportWarning,
  'IndentationError': IndentationError,
  'IndexError': IndexError,
  'KeyError': KeyError,
  'KeyboardInterrupt': KeyboardInterrupt,
  'LookupError': LookupError,
  'MemoryError': MemoryError,
  'NameError': NameError,
  'None': None,
  'NotImplemented': NotImplemented,
  'NotImplementedError': NotImplementedError,
  'OSError': OSError,
  'OverflowError': OverflowError,
  'PendingDeprecationWarning': PendingDeprecationWarning,
  'ReferenceError': ReferenceError,
  'RuntimeError': RuntimeError,
  'RuntimeWarning': RuntimeWarning,
  'StandardError': StandardError,
  'StopIteration': StopIteration,
  'SyntaxError': SyntaxError,
  'SyntaxWarning': SyntaxWarning,
  'SystemError': SystemError,
  'SystemExit': SystemExit,
  'TabError': TabError,
  'True': True,
  'TypeError': TypeError,
  'UnboundLocalError': UnboundLocalError,
  'UnicodeDecodeError': UnicodeDecodeError,
  'UnicodeEncodeError': UnicodeEncodeError,
  'UnicodeError': UnicodeError,
  'UnicodeTranslateError': UnicodeTranslateError,
  'UnicodeWarning': UnicodeWarning,
  'UserWarning': UserWarning,
  'ValueError': ValueError,
  'Warning': Warning,
  'WindowsError': WindowsError,
  'ZeroDivisionError': ZeroDivisionError,
  '__IPYTHON__': True,
  '__debug__': True,
  '__doc__': "Built-in functions, exceptions, and other objects.\n\nNoteworth
```

```
y: None is the `nil' object; Ellipsis represents `...' in slices.",
  '__import__': <function __import__>,
  '__name__': '__builtin__',
  '__package__': None,
  'abs': <function abs>,
  'all': <function all>,
  'any': <function any>,
  'apply': <function apply>,
  'basestring': basestring,
  'bin': <function bin>,
  'bool': bool,
  'buffer': buffer,
  'bytearray': bytearray,
  'bytes': str,
  'callable': <function callable>,
  'chr': <function chr>,
  'classmethod': classmethod,
  'cmp': <function cmp>,
  'coerce': <function coerce>,
  'compile': <function compile>,
  'complex': complex,
  'copyright': Copyright (c) 2001-2016 Python Software Foundation.
  All Rights Reserved.

  Copyright (c) 2000 BeOpen.com.
  All Rights Reserved.

  Copyright (c) 1995-2001 Corporation for National Research Initiatives.
  All Rights Reserved.

  Copyright (c) 1991-1995 Stichting Mathematisch Centrum, Amsterdam.
  All Rights Reserved.,
  'credits':     Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast
 of thousands
      for supporting Python development.  See www.python.org for more informa
tion.,
  'delattr': <function delattr>,
  'dict': dict,
  'dir': <function dir>,
  'divmod': <function divmod>,
  'dreload': <function IPython.lib.deepreload._dreload>,
  'enumerate': enumerate,
  'eval': <function eval>,
  'execfile': <function execfile>,
  'file': file,
  'filter': <function filter>,
  'float': float,
  'format': <function format>,
  'frozenset': frozenset,
  'get_ipython': <bound method ZMQInteractiveShell.get_ipython of <ipykernel.
zmqshell.ZMQInteractiveShell object at 0x03479A70>>,
  'getattr': <function getattr>,
  'globals': <function globals>,
  'hasattr': <function hasattr>,
  'hash': <function hash>,
  'help': Type help() for interactive help, or help(object) for help about ob
ject.,
  'hex': <function hex>,
  'id': <function id>,
  'input': <function ipykernel.ipkernel.<lambda>>,
  'int': int,
  'intern': <function intern>,
  'isinstance': <function isinstance>,
  'issubclass': <function issubclass>,
  'iter': <function iter>,
  'len': <function len>,
  'license': Type license() to see the full license text,
  'list': list,
  'locals': <function locals>,
  'long': long,
  'map': <function map>,
  'max': <function max>,
  'memoryview': memoryview,
  'min': <function min>,
  'next': <function next>,
  'object': object,
  'oct': <function oct>,
  'open': <function open>,
  'ord': <function ord>,
  'pow': <function pow>,
  'print': <function print>,
  'property': property,
  'range': <function range>,
  'raw_input': <bound method IPythonKernel.raw_input of <ipykernel.ipkernel.I
PythonKernel object at 0x03479B70>>,
  'reduce': <function reduce>,
  'reload': <function reload>,
  'repr': <function repr>,
  'reversed': reversed,
  'round': <function round>,
  'set': set,
  'setattr': <function setattr>,
  'slice': slice,
  'sorted': <function sorted>,
  'staticmethod': staticmethod,
  'str': str,
  'sum': <function sum>,
  'super': super,
  'tuple': tuple,
  'type': type,
  'unichr': <function unichr>,
  'unicode': unicode,
  'vars': <function vars>,
  'xrange': xrange,
  'zip': <function zip>},
 'b': 'sparrow',
 'birds': ['parrot', 'pigeon', 'sparrow'],
 'x': 7,
 'y': 10}
```

when a string is passed to exec,eval(), or execfile(), parser first compiles to create bytecode.

To remove this redundant process every time, compile will create precompiled bytecode, which can be used everytime, till the code is not changed

compile (str, filename, kind) function a string compiled into byte code, str is the string to be compiled, the filename is to define the string variable file, the kind parameter specifies the type of code is compiled

- ' Single 'refers to a single statement,
- ' exec 'means more than one statement,
- ' eval 'means an expression.

compile () function returns a code object, the object, of course, can also be passed to the eval () function and the exec statement to perform, for example, :

```
In [37]: str = "for i in range (0,10): print i,"
         c = compile (str,'', 'exec') # compiled to byte code object
         exec c # execution

         0 1 2 3 4 5 6 7 8 9
```

```
In [38]: str2 = "for i in range (0,10): print i,"
         c2 = compile (str2,'', 'eval') # eval() can't  execute statements
         eval(c2)

           File "<string>", line 1
             for i in range (0,10): print i,
               ^
         SyntaxError: invalid syntax
```

```
In [57]: str2 = "[0,1,2,3,4,5,6,7,8,9]+ [99,88,77]"
         c2 = compile (str2,'', 'eval') # eval() can execute expressions
         eval(c2)

Out[57]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 99, 88, 77]
```

```
In [58]: print c2, type(c2) # code object

         <code object <module> at 0384DCC8, file "", line 1> <type 'code'>
```

```
In [59]: print dir(c2)

         ['__class__', '__cmp__', '__delattr__', '__doc__', '__eq__', '__format__', '_
         _ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt
         __', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setat
         tr__', '__sizeof__', '__str__', '__subclasshook__', 'co_argcount', 'co_cellva
         rs', 'co_code', 'co_consts', 'co_filename', 'co_firstlineno', 'co_flags', 'co
         _freevars', 'co_lnotab', 'co_name', 'co_names', 'co_nlocals', 'co_stacksize',
          'co_varnames']
```

py_compile - It is a module to create bytecode file, *.pyc*

In [64]:
```
import py_compile

py_compile.compile('fileName.py')
```

In [65]:
```
os.listdir('.')
```

Out[65]:
```
['.ipynb_checkpoints',
 'desktop.ini',
 'fibGenerator.py',
 'fibGenerator.pyc',
 'fileName.py',
 'fileName.pyc',
 'first',
 'myfile.pyw',
 'myNewFile.tsf',
 'myNewFolder',
 'newFolder',
 'newScript.py',
 'newScript.pyc',
 'python - Basic Level Module.ipynb',
 'python+-+Basic+Level+Module.html']
```

In Python 2.x, input(...) is equivalent to eval(raw_input(...))

In Python 3.x, raw_input() was renamed input()

7. Iterables, Iterators and Generators

7.1 Iterables

- Objects, over which we can iterate, are called Iterables.
- String, List, tuple, set, dictionary objectes are iterables.
- int, float, complex, boolean objects are NOT iterables.

In [1184]:
```
for i in range(7):
    print 'Hey ', i
```

```
Hey  0
Hey  1
Hey  2
Hey  3
Hey  4
Hey  5
Hey  6
```

In [1185]:
```
for i in [12, 23, 34, 54, 56]:
    print i,
```

```
12 23 34 54 56
```

```
In [1186]: print [char for char in 'Python Programming']
```

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

```
In [1187]: for key in {'a': 'Apple', 'b': 'Ball'}:
               print key,
```

```
a b
```

NOTE: The default iterator for dictionary is keys()

```
In [1188]: for item in {'a': 'Apple', 'b': 'Ball'}.items():
               print item,
```

```
('a', 'Apple') ('b', 'Ball')
```

Also, iterators can be used in other ways

```
In [1189]: '-'.join(['Date', 'Month', 'Year'])
```

```
Out[1189]: 'Date-Month-Year'
```

```
In [1190]: '-'.join({'Date':8, 'Month':4, 'Year': 2016})
```

```
Out[1190]: 'Date-Year-Month'
```

```
In [1191]: num = {'Date':8, 'Month':4, 'Year': 2016}
           print num.values()

           '-'.join(str(num.values()))
```

```
[8, 2016, 4]
```

```
Out[1191]: '[-8-,- -2-0-1-6-,- -4-]'
```

Assignment : using join method of strings try to display as '8-4-2016' if the input dictionary is {'Date':8, 'Month':4, 'Year': 2016}

```
In [1195]: list('Programming')
```

```
Out[1195]: ['P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g']
```

```
In [1196]: list({'Date':8, 'Month':4, 'Year': 2016})
```

```
Out[1196]: ['Date', 'Year', 'Month']
```

7.2 Iteration (Iter) protocol

iter() - takes an iterable object and retuns an iterator

next() - method call to retun elements from the iterator. Results in StopIterator error, if the elements are not present

```
In [1197]: range(9)

Out[1197]: [0, 1, 2, 3, 4, 5, 6, 7, 8]
```

```
In [1198]: xrange(9)

Out[1198]: xrange(9)
```

```
In [1199]: a = xrange(9)
```

```
In [1200]: print a

xrange(9)
```

```
In [1201]: for i in xrange(9):
               print i,

           print "\n"
           for i in range(9):
               print i,

0 1 2 3 4 5 6 7 8

0 1 2 3 4 5 6 7 8
```

```
In [1202]: li = iter([12,23,34])  # list iterator         # iter() builtin function - to
            create iterators
           print li, type(li)

<listiterator object at 0x03EECA10> <type 'listiterator'>
```

```
In [1203]: l = [12, 23, 45, 56]
           print l, type(l)

           li = iter(l)
           print li, type(li)

[12, 23, 45, 56] <type 'list'>
<listiterator object at 0x03EEC630> <type 'listiterator'>
```

```
In [1204]: print dir(l)
```

```
['__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 [1205]: print dir(li)
```

```
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__
hash__', '__init__', '__iter__', '__length_hint__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subc
lasshook__', 'next']
```

```
In [1206]: print li.next()
```

```
12
```

```
In [1207]: print li.next()
```

```
23
```

```
In [1208]: print li.next(), li.next()
```

```
45 56
```

```
In [1209]: print li.next()    # As there are no more values in it
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-1209-689c4d344527> in <module>()
----> 1 print li.next()    # As there are no more values in it

StopIteration:
```

```
In [1210]: t = (12, 23, 45, 56)
           print t, type(t)

           ti = iter(t)        # tuple iterator
           print ti, type(ti)
```

```
(12, 23, 45, 56) <type 'tuple'>
<tupleiterator object at 0x03EECD90> <type 'tupleiterator'>
```

```
In [1211]: print dir(ti)
```

```
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__
hash__', '__init__', '__iter__', '__length_hint__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subc
lasshook__', 'next']
```

```
In [1212]: ti.next()

Out[1212]: 12
```

```
In [1213]: print ti.next(), ti.next(), ti.next()

           23 45 56
```

```
In [1214]: print ti.next()

           ---------------------------------------------------------------------------
           StopIteration                             Traceback (most recent call last)
           <ipython-input-1214-6ce46c9248fd> in <module>()
           ----> 1 print ti.next()

           StopIteration:
```

```
In [1215]: s = {12,23,34}
            print s, type(s)

            si = iter(s)         # set iterator
            print si, type(si)

           set([34, 12, 23]) <type 'set'>
           <setiterator object at 0x03EF3558> <type 'setiterator'>
```

```
In [1216]: print dir(si)

           ['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__
           hash__', '__init__', '__iter__', '__length_hint__', '__new__', '__reduce__',
            '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subc
           lasshook__', 'next']
```

```
In [1217]: print si.next(), si.next(), si.next(), si.next()   # exception is for the 4th
            call

           34 12 23

           ---------------------------------------------------------------------------
           StopIteration                             Traceback (most recent call last)
           <ipython-input-1217-0ff75836b066> in <module>()
           ----> 1 print si.next(), si.next(), si.next(), si.next()   # exception is for
            the 4th call

           StopIteration:
```

Assignment : Try the iter() protocol for frozenset

In [1218]:
```
d = {'a':12, 'b':23, 'c':34}
print d, type(d)

di = iter(d)          # dictionary iterator
print di, type(di)
```

```
 {'a': 12, 'c': 34, 'b': 23} <type 'dict'>
<dictionary-keyiterator object at 0x03E45DE0> <type 'dictionary-keyiterator'>
```

In [1219]:
```
print dir(di)
```

```
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__
hash__', '__init__', '__iter__', '__length_hint__', '__new__', '__reduce__',
 '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subc
lasshook__', 'next']
```

In [1220]:
```
print di.next()
```

```
a
```

In [1221]:
```
print di.next(), di.next(), di.next()
```

```
c b

---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-1221-35f26283dd3e> in <module>()
----> 1 print di.next(), di.next(), di.next()

StopIteration:
```

Assignment : Try to get the dictionary pair in dictionary iterator object

7.3 Generators

- It simplifies the creation of iterators
- It is a function that returns a sequence of results, rather than a single result.
- If a function uses the 'yield' keyword, it creates a generator object.
- yield is different from return.

Interview Question : what is the difference between iterator and generator?

Interview Question : what is the difference between funnction and generator?

```
In [1222]: def count(n):
               print "Stating to count!"
               i = 0
               while i<n:
                   yield i
                   i+=1
               print '$', i
               #return  i   # PEP8 strongly discourages usage of yield and retun, in same
            function
```

```
In [1223]: c = count(6)
```

```
In [1224]: print c

           <generator object count at 0x03EF3AF8>
```

```
In [1225]: print c.next()    # execution starts when .next() is given   # It stores the s
           tate,  after execution

           Stating to count!
           0
```

```
In [1226]: c.next()

Out[1226]: 1
```

```
In [1227]: print c.next(), c.next(), c.next()

           2 3 4
```

```
In [1228]: c.next()

Out[1228]: 5
```

```
In [1229]: c.next()         # because there are no more values

           $ 6

           ---------------------------------------------------------------------------
           StopIteration                             Traceback (most recent call last)
           <ipython-input-1229-69882ed18d78> in <module>()
           ----> 1 c.next()         # because there are no more values

           StopIteration:
```

This function doesn't get executed when the function call is made; but executed when the next() method call is done.

```
In [1230]: def foo():
               print "Start the function!"
               for i in range(3):
                   print "before yield", i
                   yield i
                   print "after yield", i
               print "end of function "
```

```
In [1231]: f = foo()
```

```
In [1232]: type(f)

Out[1232]: generator
```

```
In [1233]: f.next()

           Start the function!
           before yield 0

Out[1233]: 0
```

```
In [1234]: f.next()

           after yield 0
           before yield 1

Out[1234]: 1
```

```
In [1235]: f.next()

           after yield 1
           before yield 2

Out[1235]: 2
```

```
In [1236]: f.next()

           after yield 2
           end of function

           ---------------------------------------------------------------------------
           StopIteration                             Traceback (most recent call last)
           <ipython-input-1236-c3e65e5362fb> in <module>()
           ----> 1 f.next()

           StopIteration:
```

Interview Question : What is the difference between yield and return?

yield will halt the execution, untill the next next() method is encountered. Where as *return* will return the result at only, and won't go back to the function

Generator function terminates by calling either *return* or by raising StopIteration error.

It is not recommended to place both yield and return for the same function.

```
In [1239]: def yrange(finalValue, initialValue = 0,  step = 1):
                i = 0
                while i< finalValue:
                    yield i
                    i += step
```

```
In [1240]: for i in yrange(3):
                print i

            0
            1
            2
```

```
In [1241]: y = yrange(3)
```

```
In [1242]: print y, type(y)

            <generator object yrange at 0x03EF3FA8> <type 'generator'>
```

```
In [1243]: y.next()

Out[1243]: 0
```

```
In [1244]: for i in y:
                print i        # prints the remaining elements in the generator object

            1
            2
```

```
In [1245]: def integers():
                """Infinite sequence of integers."""
                i = 1
                while True:
                    yield i
                    i = i + 1

            def squares():
                for i in integers():
                    yield i * i

            def take(n, seq):
                """Returns first n values from the given sequence."""
                seq = iter(seq)
                result = []
                try:
                    for i in range(n):
                        result.append(seq.next())
                except StopIteration:
                    pass
                return result

            print take(5, squares()) # prints [1, 4, 9, 16, 25]
```

```
[1, 4, 9, 16, 25]
```

```
In [1246]: #!/usr/bin/python
            # Purpose: To make a Fibonacci generator.
            # fibGenerator.py

            def fibonacci(max):
                n, a, b = 0, 0, 1
                while n < max:
                    yield a
                    a, b = b, a + b
                    n = n + 1


            if __name__ == '__main__':     # This condition gets executed, only if the pyt
            hon script is directly executed
                fib10 = fibonacci(10)
                for i in fib10:
                    print i,
```

```
0 1 1 2 3 5 8 13 21 34
```

```
In [277]: import os; os.listdir(os.getcwd())

Out[277]: ['.ipynb_checkpoints',
           'desktop.ini',
           'fibGenerator.py',
           'first',
           'myfile.pyw',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'newScript.py',
           'newScript.pyc',
           'python - Basic Level Module.ipynb',
           'Untitled.ipynb']

In [278]: from fibGenerator import fibonacci

In [279]: fibonacci

Out[279]: <function fibGenerator.fibonacci>

In [280]: type(fibonacci)

Out[280]: function

In [281]: fib = fibonacci(10)

In [282]: print fib, type(fib)

          <generator object fibonacci at 0x0355C3C8> <type 'generator'>

In [283]: fib.next()

Out[283]: 0

In [284]: for i in fib:
              print i,

          1 1 2 3 5 8 13 21 34

In [285]: reload(fibonacci)

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-285-e616c20e4b49> in <module>()
          ----> 1 reload(fibonacci)

          TypeError: reload() argument must be module
```

In [286]:
```python
#del fibonacci

import fibGenerator

reload(fibGenerator)   # .pyc file will be recreated
```
Out[286]:
```
<module 'fibGenerator' from 'fibGenerator.pyc'>
```

In [287]:
```python
f = fibGenerator.fibonacci(10)

for i in f:
    print i,
```
```
0 1 1 2 3 5 8 13 21 34
```

7.4 Generator Expressions

- tuple comprehension
  - It is generator version of list comprehension.
- List comprehension creates a sequence that contains the resulting data. Generator expression creates a generator that knows how to produce data on demand.
- Generator Expression (GE) improves performance and memory usage
- GE creates objects, which can't be indexed.

```
syntax:
    (expression for item1 in iterable1
                for item2 in iterable2
                for item3 in iterable3
                .
                .
                .
                for itemN in iterableN
                if condition)
```
`

In [1247]:
```python
b = (10*i for i in [1,2,3,4])
```

In [1248]:
```python
print b, type(b)
```
```
<generator object <genexpr> at 0x03EF35F8> <type 'generator'>
```

In [1249]:
```python
b.next()
```
Out[1249]:
```
10
```

In [1250]:
```python
b.next()
```
Out[1250]:
```
20
```

```
In [1251]: b.next(), b.next(), b.next()
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-1251-10d82aa87e61> in <module>()
----> 1 b.next(), b.next(), b.next()

StopIteration:
```

```
In [1252]: c = list(b)   # Generator expression to list conversion

           print c, type(c)
```

```
[] <type 'list'>
```

```
In [1253]: b = (10*i for i in [1,2,3,4])
           c = list(b)   # Generator expression to list conversion

           print c, type(c)
```

```
[10, 20, 30, 40] <type 'list'>
```

Interview Question : what is the result of this :

```
    sum(i*i for i in range(10))
```

```
In [1262]: sum(i*i for i in range(10))   # sum()  - builtin function to result the summat
           ion
```

```
Out[1262]: 285
```

```
In [1263]: sum([i*i for i in range(10)])
```

```
Out[1263]: 285
```

Assignment : Try to evaluate the decimal and complex numbers in sum() function

Assignment : Try to mimic the sum() function functionality using reduce function

7.5 Itertools

- chain - chains multiple iterators together
- izip - iterable version of zip
- product - computes the cartesian product of input iterables

```
        product(A,B) is same as ((x,y) for x in A for y in B)
```

```
In [1264]: import itertools
In [1265]: li1 = iter([1,2,3])
           li2 = iter([4,5,6])
           a = itertools.chain(li1, li2)
           print type(a)
           print a

           <type 'itertools.chain'>
           <itertools.chain object at 0x03F1F6F0>
```

```
In [1266]: list(a)

Out[1266]: [1, 2, 3, 4, 5, 6]
```

```
In [1267]: list(itertools.chain([1,2,3], [4,5,6]))

Out[1267]: [1, 2, 3, 4, 5, 6]
```

```
In [1268]: list(itertools.chain([[1,2,3], [4,5,6]]))

Out[1268]: [[1, 2, 3], [4, 5, 6]]
```

```
In [1269]: list(itertools.chain([[1,2,3], [4,5,6]], [99, 79, 69]))

Out[1269]: [[1, 2, 3], [4, 5, 6], 99, 79, 69]
```

```
In [1270]: list(itertools.chain(['ABC', 'DEF']))

Out[1270]: ['ABC', 'DEF']
```

```
In [1271]: list(itertools.chain.from_iterable(['ABC', 'DEF']))

Out[1271]: ['A', 'B', 'C', 'D', 'E', 'F']
```

```
In [1272]: list(itertools.chain.from_iterable([[1,2,3], [4,5,6]], [99, 79, 69]))

           ---------------------------------------------------------------------------
           TypeError                                 Traceback (most recent call last)
           <ipython-input-1272-8061948b2310> in <module>()
           ----> 1 list(itertools.chain.from_iterable([[1,2,3], [4,5,6]], [99, 79, 69]))

           TypeError: from_iterable() takes exactly one argument (2 given)
```

Interview Question : How to convert a multi-dimensional list to a flat list

```
input: [[1,2,3], [4,5,6], [99, 79, 69]]

output: [1, 2, 3, 4, 5, 6, 99, 79, 69]
```

```
In [1273]: list(itertools.chain.from_iterable([[1,2,3], [4,5,6], [99, 79, 69]]))

Out[1273]: [1, 2, 3, 4, 5, 6, 99, 79, 69]
```

Assignment : Try to do multi-dimensional list to single dimensional list, or list flattening, using itertools

In [1274]:
```
for x, y in itertools.izip(["a", "b", "c"], [1, 2, 3]):
        print x,y
```
```
a 1
b 2
c 3
```

In [1275]:
```
list(itertools.izip_longest('abcd', 'ABCD', fillvalue='-'))
```
Out[1275]:
```
[('a', 'A'), ('b', 'B'), ('c', 'C'), ('d', 'D')]
```

In [1276]:
```
list(itertools.izip_longest('abcd', 'AB', fillvalue='-'))
```
Out[1276]:
```
[('a', 'A'), ('b', 'B'), ('c', '-'), ('d', '-')]
```

In [1277]:
```
list(itertools.izip_longest('ab', 'ABCD', fillvalue='-'))
```
Out[1277]:
```
[('a', 'A'), ('b', 'B'), ('-', 'C'), ('-', 'D')]
```

In [1278]:
```
list(itertools.izip_longest('cd', 'ABCD', fillvalue='-'))
```
Out[1278]:
```
[('c', 'A'), ('d', 'B'), ('-', 'C'), ('-', 'D')]
```

In [1279]:
```
print list(itertools.product([1,2,3], repeat = 2))
```
```
[(1, 1), (1, 2), (1, 3), (2, 1), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
```

In [1280]:
```
print list(itertools.product([1,2,3], repeat = 0))
```
```
[()]
```

In [1281]:
```
print list(itertools.product([1,2,3], repeat = 1))
```
```
[(1,), (2,), (3,)]
```

In [1282]:
```
print list(itertools.product([1,2,3], repeat = 3))
```
```
[(1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 2, 1), (1, 2, 2), (1, 2, 3), (1, 3, 1),
 (1, 3, 2), (1, 3, 3), (2, 1, 1), (2, 1, 2), (2, 1, 3), (2, 2, 1), (2, 2, 2),
 (2, 2, 3), (2, 3, 1), (2, 3, 2), (2, 3, 3), (3, 1, 1), (3, 1, 2), (3, 1, 3),
 (3, 2, 1), (3, 2, 2), (3, 2, 3), (3, 3, 1), (3, 3, 2), (3, 3, 3)]
```

In [1283]:
```
print list(itertools.product([1,2,3],[3,4]))
```
```
[(1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (3, 4)]
```

In [1284]:
```
s = [[1,2,3],[3,4,5]]
print list(itertools.product(*s))    # UNPACKING
```
```
[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)]
```

In [1285]:
```
print list(itertools.product(s))
```
```
[([1, 2, 3],), ([3, 4, 5],)]
```

In [1286]:
```
s = [(1,2,3),[3,4,5]]   # non-homegeneous list
print list(itertools.product(*s))
```
```
[(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 3), (3, 4), (3, 5)]
```

In [1287]:
```
s = [(1,2,(45,78,9),3),[3,4,[33, (44,)],5]]  # multi-dimensional list
list(itertools.product(*s))
```
Out[1287]:
```
[(1, 3),
 (1, 4),
 (1, [33, (44,)]),
 (1, 5),
 (2, 3),
 (2, 4),
 (2, [33, (44,)]),
 (2, 5),
 ((45, 78, 9), 3),
 ((45, 78, 9), 4),
 ((45, 78, 9), [33, (44,)]),
 ((45, 78, 9), 5),
 (3, 3),
 (3, 4),
 (3, [33, (44,)]),
 (3, 5)]
```

In [1288]:
```
t = ((1,2,(45,78,9),3),[3,4,[33, 44],5])  # multi-dimensional tuple
print list(itertools.product(*t))         # displaying as a list
```
```
[(1, 3), (1, 4), (1, [33, 44]), (1, 5), (2, 3), (2, 4), (2, [33, 44]), (2,
 5), ((45, 78, 9), 3), ((45, 78, 9), 4), ((45, 78, 9), [33, 44]), ((45, 78,
 9), 5), (3, 3), (3, 4), (3, [33, 44]), (3, 5)]
```

In [1289]:
```
print tuple(itertools.product(*t))        # displaying as a tuple
```
```
((1, 3), (1, 4), (1, [33, 44]), (1, 5), (2, 3), (2, 4), (2, [33, 44]), (2,
 5), ((45, 78, 9), 3), ((45, 78, 9), 4), ((45, 78, 9), [33, 44]), ((45, 78,
 9), 5), (3, 3), (3, 4), (3, [33, 44]), (3, 5))
```

In [1290]:
```
list(itertools.permutations('AB',2))
```
Out[1290]:
```
[('A', 'B'), ('B', 'A')]
```

In [1291]:
```
list(itertools.combinations('AB',2))
```
Out[1291]:
```
[('A', 'B')]
```

In [1292]:
```
list(itertools.combinations_with_replacement('AB',2))
```
Out[1292]:
```
[('A', 'A'), ('A', 'B'), ('B', 'B')]
```

```
In [1293]: list(itertools.permutations('ABC',2))

Out[1293]: [('A', 'B'), ('A', 'C'), ('B', 'A'), ('B', 'C'), ('C', 'A'), ('C', 'B')]

In [1294]: list(itertools.permutations('ABC',3))

Out[1294]: [('A', 'B', 'C'),
            ('A', 'C', 'B'),
            ('B', 'A', 'C'),
            ('B', 'C', 'A'),
            ('C', 'A', 'B'),
            ('C', 'B', 'A')]

In [1295]: list(itertools.combinations('ABC',3))

Out[1295]: [('A', 'B', 'C')]

In [1296]: list(itertools.combinations_with_replacement('ABC',3))

Out[1296]: [('A', 'A', 'A'),
            ('A', 'A', 'B'),
            ('A', 'A', 'C'),
            ('A', 'B', 'B'),
            ('A', 'B', 'C'),
            ('A', 'C', 'C'),
            ('B', 'B', 'B'),
            ('B', 'B', 'C'),
            ('B', 'C', 'C'),
            ('C', 'C', 'C')]

In [1297]: list(itertools.compress('ABCDEF', [1,0,1,0,1,1]))

Out[1297]: ['A', 'C', 'E', 'F']

In [1298]: list(itertools.compress('ABCDEF', [1,0,1,0,1,1,1]))

Out[1298]: ['A', 'C', 'E', 'F']

In [1299]: list(itertools.compress('ABCDEF', [1,1,1,1,1,1,1]))

Out[1299]: ['A', 'B', 'C', 'D', 'E', 'F']

In [1300]: list(itertools.compress('ABCDEF', [0,0,0,0,0,0,0]))

Out[1300]: []

In [1301]: list(itertools.compress('ABCDEF', [1,0,0,0,0,0,0,1, 1, 1, 1, 1]))

Out[1301]: ['A']
```

Exceptions

Almost all programming languages, except shell scripting and some scripting languages, possess exception handling capabilities.

There are two kinds of errors in Python.

1. error code - If something went wrong, the resulting error code is -1 to indicate the failure of a call.
2. Exception - Used to handle exceptional cases.

In Python, the errors are handled by the interpreter by raising an exception and allowing that exception to be handled.

Exceptions indicate errors and break out of the normal control flow of a program. An exception is raised using raise statement.

```
Syntax:
    try:
        logic
        ...
    except <ExceptionName1>, <alias identifier>:
        logic to handle that exception
        ...
    except <ExceptionName2> as <alias identifier>:
        logic to handle that exception.
        This logic gets executed, if error is not covered
        in ExceptionName1 exception
        ...
    else:
        logic to execute if
        there is no exception
        ...
    finally:
        logic to execute either
        if exception occurs are not
        ...
```

Note : *try* and *except* are mandatory blocks. And, *else* and *finally* are optional blocks.

```
In [1302]: result = 21/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-1302-123e673b3321> in <module>()
----> 1 result = 21/0

ZeroDivisionError: integer division or modulo by zero
```

In [1303]:
```
try:
    result = 21/0
except:
    print 'An error occurred!'
```

```
An error occurred!
```

This code handles all exceptions. But, we should know the error to do corresponding action

In [1304]:
```
try:
    result = 21/0
except:
    print 'An error occurred'
    print 'The error is %s'%(ex)
```

```
An error occurred

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1304-e878884284c3> in <module>()
      3 except:
      4     print 'An error occurred'
----> 5     print 'The error is %s'%(ex)
      6 

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

Here, the exception was resulted in the exception block

In [1305]:
```
try:
    result = (1+2.3)/(2*4*0)
except:
    print 'An error occurred'
    try:
        print 'The error is %s'%(ex)
    except:
        print 'variable "ex" is not defined'
```

```
An error occurred
variable "ex" is not defined
```

In [1306]:
```
import exceptions
try:
        result = (1+2.3)/(2*4*0)
except exceptions.Exception, ex:
        print "The error is ", ex
```

```
The error is  float division by zero
```

Ensure that all the exceptions are handled in the code

In [1307]:
```
try:
        result = (1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", repr(ex)
        print "The error is ", ex
```

```
The error is  ZeroDivisionError('float division by zero',)
The error is  float division by zero
```

In [1308]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", ex
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1308-d6eb5a1e6e41> in <module>()
      1 try:
----> 2         result = (w)*(1+2.3)/(2*4*0)
      3 except ZeroDivisionError, ex:  # better handling of exception
      4         print "The error is ", ex

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

Interview Question : If there are two errors, suppose ZeroDivisionError and NameError, which error will be resulted

In [1309]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The error is ", ex
        print "The error is ", repr(ex)
except NameError, ex:
        print "The error is ", ex
        print "The error is ", repr(ex)
```

```
The error is  name 'w' is not defined
The error is  NameError("name 'w' is not defined",)
```

There should be a except to accept any unknown exception

In [1310]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
        print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
```

```
The NameError is  name 'w' is not defined
```

```
control flow:
    when there is no exception:
        try -> else -> finally
    when there is exception:
        try -> except -> finally
```

In [1311]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
The NameError is  name 'w' is not defined
Completed the try exception block
```

In [1312]:
```
try:
        result = (9)*(1+2.3)/(2*4*10)
except ZeroDivisionError, ex:  # better handling of exception
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
try block executed successfully
The result is  0.37125
Completed the try exception block
```

As 'finally' gets executed in either case, it is seldom used. finally is generally used for cleanup action

In [1313]:
```
try:
    x = float(raw_input("Your number: "))
    inverse = 1.0 / x
finally:
    print("There may or may not have been an exception.")
print "The inverse: ", inverse
```

```
Your number: thousand
There may or may not have been an exception.

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-1313-cd0c760fc506> in <module>()
      1 try:
----> 2     x = float(raw_input("Your number: "))
      3     inverse = 1.0 / x
      4 finally:
      5     print("There may or may not have been an exception.")

ValueError: could not convert string to float: thousand
```

Here, exception is throwed, as except block is missing

In [1314]:
```
try:
    x = float(raw_input("Your number: "))
    inverse = 1.0 / x
except exceptions.Exception, ex:
        print "The error is ", ex
finally:
    print("There may or may not have been an exception.")
print "The inverse: ", inverse
```

```
Your number: 1000
There may or may not have been an exception.
The inverse:  0.001
```

Built-in exceptions:

```
- Exception                                 - root of all exceptions
    - SystemExit                            - generated by sys.exit()
    - StopIteration                         - Raised to stop iteration
    - StandardError                         - Base of all built-in exceptions
        - ArithmeticError                   - Base for arithmetic exceptions
            - FloatingPointError            - faliture of a floating-point operation
            - OverflowError                 - Arithmetic overflow
            - ZeroDivisionError             - Division or modulus operation with 0
        - AssertionError                    - Raised by assert statement
        - AttributeError                    - Raised when an attribute name is inval
id
        - EnvironmentError                  - Errors that occur externally to python

            - IOError                       - I/O or file-related error
            - OSError                       - Operating System error
        - EOFError                          - Raised when end of file is reached
        - ImportError                       - Failure of import statement
        - KeyboardInterrupt                 - Generated by interrupt key (ctrl+C or
 ctrl+D)
        - LookupError                       - Indexing and key errors
            - IndexError                    - Out-of-range sequence offset
            - KeyError                      - Non-existent dictionary key
        - MemoryError                       - Out of memory error
        - NameError                         - Failure to find a local or global name
            - UnboundLocalError             - unbound local variable
        - ReferenceError                    - Weak reference used after referent des
troyed
        - RuntimeError                      - A generic catch-all error
            - NotImplementedError           - unimplemented feature
        - SyntaxError                       - Parsing Error
            - IndentationError              - Indentation error
            - TabError                      - Inconsistent tab usage(generated with
 -tt option)
        - SystemError                       - Non-fatal system error in interpreter
        - TypeError                         - Passing an inappropriate type to an op
eration
        - ValueError                        - Invalid type
            - UnicodeError                  - unicode error
                - UnicodeDecodeError        - unicode decoding error
                - UnicodeEncodeError        - unicode encoding error
                - UnicodeTranslateError     - unicode translation error
```

In [1315]:
```
try:
        result = (w)*(1+2.3)/(2*4*0)   # This statement contains both NameErro
r and ZeroDivisionError
except NameError, ex:
    print "The NameError is ", ex
except ZeroDivisionError, ex:      # better handling of exception
        print "The ZeroDivisionError is ", ex
except StandardError, ex:
    print "All Standard Errors are handled here"
    print "The error is ",ex
except exceptions.Exception, ex:
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
The NameError is  name 'w' is not defined
Completed the try exception block
```

In the above example, notice that NameError caught the exception, rather than StandardError, due to its placement preceeding StandardError exception.

In [1316]:
```
import exceptions
try:
        result = (12)*(1+2.3)/(2*4*0)   # This statement contains ONLY ZeroDiv
isionError
except exceptions.Exception, ex:    # It handles all exceptions
   print "ALL the errors are handled here"
   print "The other error is ", ex
except StandardError, ex:           # handles only Standard Errors
    print "All Standard Errors are handled here"
    print "The error is ",ex
except ZeroDivisionError, ex:
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
ALL the errors are handled here
The other error is  float division by zero
Completed the try exception block
```

In the above scenario, notice that Exception method of exceptions module is handling all the exceptions, rather than the StandardError exception, due to its heirarchy.

In [1317]:
```
import exceptions
try:
        result = (12)*(1+2.3)/(2*4*0)   # This statement contains ONLY ZeroDiv
isionError
except StandardError, ex:           # handles only Standard Errors
    print "All Standard Errors are handled here"
    print "The error is ",ex
except ZeroDivisionError, ex:
        print "The ZeroDivisionError is ", ex
except NameError, ex:
    print "The NameError is ", ex
except exceptions.Exception, ex:    # It handles all exceptions
        print "ALL the errors are handled here"
        print "The other error is ", ex
else:
    print "try block executed successfully"
    print "The result is ", result
finally:
    print "Completed the try exception block"
```

```
All Standard Errors are handled here
The error is  float division by zero
Completed the try exception block
```

In [1319]:
```
### !/usr/bin/python
# exceptionsExample1.py
import exceptions

try:
  a = int(input('please enter your number:'))
except ValueError:
  print "we should not enter zero"         # It is a logic error. Logical erro
rs can't be handled
else:
  print "the number a:%d" %(a)
finally:
  print "finally clause"
```

```
please enter your number:'thousand'
we should not enter zero
finally clause
```

In [1320]:
```
#!/usr/bin/python
# exceptionsExample1.py
import exceptions

try:
  a = int(input('please enter your number:'))
except ValueError as ex:
  print 'Please enter numbers only'
  print ex
else:
  print "the number a:%d" %(a)
finally:
  print "finally clause"
```

```
please enter your number:'thousand'
Please enter numbers only
invalid literal for int() with base 10: 'thousand'
finally clause
```

In [1321]:
```
#!/usr/bin/python
# exceptionsExample2.py

try:
    age = int(input('please enter your age:'))
    print "my age is:", age
except NameError,error:
    print "please enter your age in numeric"
except:
    print 'someElse error'



print "----- report --------"
print "We got the following error : %s" %(error)
           # identifier 'error' is being used outside of the try-except blocks
```

```
please enter your age:36
my age is: 36
----- report --------

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1321-35c972bc047c> in <module>()
     12
     13 print "----- report --------"
---> 14 print "We got the following error : %s" %(error)
     15            # identifier 'error' is being used outside of the try-exce
pt blocks

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

In [1322]:
```
#!/usr/bin/python
import sys

try:
    value1 = input("Please enter the value1:")
    value2 = input("Please enter the value2:")
except NameError,error:
    print "Please enter only numbers !!!"
print "ERROR: %s" %(error)
```

```
Please enter the value1:12
Please enter the value2:'twelve'

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1322-710987f9aacf> in <module>()
      7 except NameError,error:
      8  print "Please enter only numbers !!!"
----> 9 print "ERROR: %s" %(error)

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

In [1323]:
```
#!/usr/bin/python

try:
    a = input('please enter your number 1:')
    b = input('please enter your number 2:')
except NameError,error:
    print "please enter numbers only \n"
    print "exception:", error
else:
    print "You enter the right number a:%d b:%d" %(a,b)
```

```
please enter your number 1:123
please enter your number 2:456
You enter the right number a:123 b:456
```

In [1324]:
```
#!/usr/bin/python

try:
    num1 = int(raw_input("please enter a number1:"))
    num2 = int(raw_input("please enter a number2:"))
except ValueError:
    print "Please enter a number[0-9]"
else:
    print num1,num2
```

```
please enter a number1:python
Please enter a number[0-9]
```

In [1325]:
```
#!/usr/bin/python
import sys

try:
    value1 = input("Please enter the value1:")
    value2 = input("Please enter the value2:")
except NameError:
    print "Please enter only numbers !!!"
    sys.exit(1)
except exceptions.Exception, ex:    # It handles all exceptions
    print "ALL the errors are handled here"
    print "The other error is ", ex
else:
    print "Division of two numbers:" , value1/value2 * 1.0
```

```
Please enter the value1:444
Please enter the value2:222
Division of two numbers: 2.0
```

In [1326]:
```
#!/usr/bin/python
import sys

try:
    value1 = input("Please enter the value1:")
    value2 = input("Please enter the value2:")
except NameError:
    print "Please enter only numbers !!!"
    sys.exit(1)
except exceptions.Exception, ex:    # It handles all exceptions
    print "ALL the errors are handled here"
    print "The other error is ", ex
else:
    print "The division of two numbers:" , value1/value2 * 1.0
```

```
Please enter the value1:123ABC
ALL the errors are handled here
The other error is  unexpected EOF while parsing (<string>, line 1)
```

In [1327]:
```
del value1, value2
```

In [1328]:
```
#!/usr/bin/python

try:
    value1 = input("Please enter your first number: ")
    value2 = input("please enter your second number: ")
except NameError:
    print "please enter numbers only \n"
    try:
        print "division of two numbers is : " , float(value1)/value2
    except (NameError,ZeroDivisionError):             # handling these two exe
cptions together
        print " \n please enter a valid number"
```

```
Please enter your first number: 999
please enter your second number: 0
```

Observe that division wasn't done as it was in except block

Multiple Exception Handling:

```
try:
    <logic>
except TypeError, ex:
    <logic to handle Type errors>
except IOError, ex:
    <logic to handle IO errors>
except NameError, ex:
    <logic to handle name errors>
except Exception, ex:
    <logic to handle any other error>

Also, single handler can catch multiple exception types.
try:
    <logic>
except (IOError, TypeError, NameError), ex:
    <logic to handle IOError, TypeError, NameError>
```

In [1329]:

```
#!/usr/bin/python

try:
    value1 = int(raw_input("please enter number 1:"))
    value2 = int(raw_input("please enter number 2:"))
except ValueError:
    print "Buddy.. its number \n"
    try:
      print "division of numbers", value1/value2
    except  ValueError,error:
      print "Buddy .. no number given .. try again .. \n"
      print "ERROR:{}".format(error)
    except ZeroDivisionError,error:
      print "Zero is not the number you would enter \n"
      print "ERROR:{}".format(error)
    except NameError,error:
      print "value is not defined"
      print "ERROR:{}".format(error)
    else:
      print "The division of numbers is success \n"
```

```
please enter number 1:13b
Buddy.. its number

division of numbers Zero is not the number you would enter

ERROR:integer division or modulo by zero
```

In [1330]:
```
#!/usr/bin/python

try:
    value1 = int(raw_input("please enter number 1:"))
    value2 = int(raw_input("please enter number 2:"))
except:
    print "Buddy.. its number \n"
    try:
      print "division of numbers", value1/value2
    except  (ValueError,ZeroDivisionError, NameError),error:
      print "Buddy .. it is either ValueError,ZeroDivisionError, NameError ..
 try again .. \n"
      print "ERROR:{}".format(error)
    else:
      print "The division of numbers is success \n"
```

```
please enter number 1:222
please enter number 2:0.0
Buddy.. its number 

division of numbers Buddy .. it is either ValueError,ZeroDivisionError, NameE
rror .. try again .. 

ERROR:integer division or modulo by zero
```

Raising exceptions

raise instance

raise class

In [1331]:
```
raise
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1331-26814ed17a01> in <module>()
----> 1 raise

TypeError: exceptions must be old-style classes or derived from BaseExceptio
n, not NoneType
```

In [1332]:
```
raise IOError
```

```
---------------------------------------------------------------------------
IOError                                   Traceback (most recent call last)
<ipython-input-1332-f6cf32b12b43> in <module>()
----> 1 raise IOError

IOError: 
```

In [1333]:
```
raise KeyboardInterrupt
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-1333-7d145351408f> in <module>()
----> 1 raise KeyboardInterrupt

KeyboardInterrupt:
```

In [1334]:
```
raise MemoryError("The memory is running out of space")
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
<ipython-input-1334-9c0834b76ffd> in <module>()
----> 1 raise MemoryError("The memory is running out of space")

MemoryError: The memory is running out of space
```

In [1335]:
```
try:
    a = int(input("Enter a positive integer: "))
    if a <= 0:
        raise ValueError("That is not a positive number!")
except ValueError as ve:
    print(ve)
```

```
Enter a positive integer: -1
That is not a positive number!
```

In [1336]:
```
#!/usr/bin/python

size = int(raw_input("please enter the size:"))

if size < 50:
    print "we have good amount of space"
if size > 50 and size < 80:
    raise UserWarning,"We are hitting 80 percentage on disk"
if size > 90:
    raise UserWarning,"we have a issue with 100 full space"
```

```
please enter the size:69

---------------------------------------------------------------------------
UserWarning                               Traceback (most recent call last)
<ipython-input-1336-8711e887e17a> in <module>()
      6     print "we have good amount of space"
      7 if size > 50 and size < 80:
----> 8     raise UserWarning,"We are hitting 80 percentage on disk"
      9 if size > 90:
     10     raise UserWarning,"we have a issue with 100 full space"

UserWarning: We are hitting 80 percentage on disk
```

Raising custom exceptions

In [1337]:
```
print 'myError'
```

```
myError
```

In [1338]:
```
'myError'
```

Out[1338]:
```
'myError'
```

In [1339]:
```
myError
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1339-8969e71fc05b> in <module>()
----> 1 myError

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

In [1340]:
```
raise NameError("This is a name error")   # python 2 and python 3
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1340-668e80d6a9a3> in <module>()
----> 1 raise NameError("This is a name error")

NameError: This is a name error
```

In [1341]:
```
raise NameError, "This is a name error" # only in python 2.x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1341-478a3e93cc64> in <module>()
----> 1 raise NameError, "This is a name error"

NameError: This is a name error
```

exception need to be a class object

In [1342]:
```
class myError(Exception(" This is myError exception")):
    pass
```

In [1343]:
```
raise myError
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-1343-99b7bfac794c> in <module>()
----> 1 raise myError

Exception: ('myError', (Exception(' This is myError exception',),), {'__modul
e__': '__main__'})
```

In [1344]:
```
import exceptions
class myError(exceptions.Exception(" This is myError exception")):
    pass
```

```
In [1345]: raise  myError
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
<ipython-input-1345-e7801eb4e931> in <module>()
----> 1 raise  myError

Exception: ('myError', (Exception(' This is myError exception',),), {'__modul
e__': '__main__'})
```

Defining custom exceptions

- All built-in exceptions are defined in terms of classes.
- To create a new exception, create a new calss definition that inherits from exceptions.Exception

```
In [1347]: #Ex1:
class Networkerror(RuntimeError):
    def __init__(self, arg):
        self.args = arg

try:
    raise Networkerror(["Bad hostname"])
except Networkerror,e:
    print e.args
```

```
('Bad hostname',)
```

```
In [1348]: #Ex2:
import exceptions
#Exception class
class NetworkError(exceptions.Exception):
        def __init__(self, errno, msg):
                self.args = (errno, msg)
                self.errno = errno
                self.errno      = msg

# Raises an exception (multiple arguments)
def error2():
        raise NetworkError(1, 'Host not found')

# Raises an exception (multiple arguments supplied as a tuple)
def error3():
        raise NetworkError, (1, 'Host not found')
```

```
In [1349]: try:
    error2() # function call
except NetworkError as ne:
    print ne
```

```
(1, 'Host not found')
```

Asserts

- used to introduce debugging code into the program.
- If the testlogic evaluates to False, assert raises an AssertionError exception with the optional data supplied.

```
Syntax:
    if not <some_test>:
        raise AssertionError(<message>)

    (or)

    assert <some_test>, <message>
```

In [1350]:
```
# Ex1:
x = 5
y = 3
assert (x < y), "x has to be smaller than y"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-1350-90ec3e3d5423> in <module>()
      2 x = 5
      3 y = 3
----> 4 assert (x < y), "x has to be smaller than y"

AssertionError: x has to be smaller than y
```

In [1351]:
```
assert (x > y), "x has to be greater than y"
```

In [1353]:
```
try:
    assert 67>89, "Condition is boolean False"
except AssertionError as ae:
    print ae
```

```
Condition is boolean False
```

Assertions are not checked when python runs in optimized mode (i.e., with -o option).

```
python -o scriptName.py
```

NOTE: assert should not be used to catch programming errors like x / 0, because Python traps such programming errors itself!.

Assert should be used for trapping user-defined constraints!

In [1355]:
```
def f():
    return 3

def testFunctionPositive():
    assert f() > 0

def testFunctionNegative():
    assert f() < 0

def testFunctionZero():
    assert f() == 0

def testFunctionEqual():
    assert f() == 3

def testFunctionNotEqual():
    assert f() != 4

# test calls
print "Positivity check: ", testFunctionPositive()
```

```
 Positivity check:  None
```

In [1356]:
```
print "Negativity check: ", testFunctionNegative()
```

```
Negativity check: 

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-1356-6880f0541be2> in <module>()
----> 1 print "Negativity check: ", testFunctionNegative()

<ipython-input-1355-6f5098ce5a43> in testFunctionNegative()
      6 
      7 def testFunctionNegative():
----> 8     assert f() < 0
      9 
     10 def testFunctionZero():

AssertionError: 
```

```
In [1357]: print "zeroness check: ", testFunctionZero()
```

```
 zeroness check:

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-1357-a399248f6f83> in <module>()
----> 1 print "zeroness check: ", testFunctionZero()

<ipython-input-1355-6f5098ce5a43> in testFunctionZero()
      9
     10 def testFunctionZero():
---> 11     assert f() == 0
     12
     13 def testFunctionEqual():

AssertionError:
```

```
In [1358]: print "value equivalence check: ", testFunctionEqual()
```

```
 value equivalence check:  None
```

```
In [1359]: try:
               print "value equivalence check: ", testFunctionNotEqual()
           except AssertionError as ae:
               print ae
```

```
value equivalence check:  None
```

NOTE: Assertions has their importance in testing. Modules like unittest, nose, robotframework deal with musch advanced assertions.

Working with files

File operation modes

r - read only

w - write only

a - appending the data

Note: If you open an existing file with 'w' mode, it's existing data get vanished.

r+ - both for read and write

a+ - both for read and append

In windows, the data is stored in binary format. Placing this 'b' doesn't effect in unix and linux.

rb - read only

wb - write only

ab - append only

ab+ - Both reading and appending data

Default file operation is **read only**.

Accessing a file

Taking a sample file, named test.txt

```
test.txt

Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
```

In [118]:
```
str1 = "Python programming is interesting\n\
It is coming with batteries, in built\n\
It means that almost every operation has a module !\n\
"
fileHandler = open('test.txt', 'wb')# creating a new file
fileHandler.write(str1)
fileHandler.close()
```

```
In [119]: import os; os.listdir(os.getcwd())

Out[119]: ['.ipynb_checkpoints',
           'desktop.ini',
           'fibGenerator.py',
           'fibGenerator.pyc',
           'fileName.py',
           'fileName.pyc',
           'first',
           'myfile.pyw',
           'myNewFile.tsf',
           'myNewFolder',
           'newFolder',
           'newScript.py',
           'newScript.pyc',
           'python - Basic Level Module.ipynb',
           'python+-+Basic+Level+Module.html',
           'test.txt']
```

```
In [120]: f = open('test.txt', 'rb')  # Opening an existing file for reading
          data1 = f.readline()        # reads one line
          f.close()
          print type(data1), data1

          <type 'str'> Python programming is interesting
```

```
In [122]: f = open('test.txt', 'rb')
          data2 = f.readlines()      # reads all lines, but results list of each line, a
          s a string
          f.close()
          print type(data2), '\n',data2

          <type 'list'>
          ['Python programming is interesting\n', 'It is coming with batteries, in buil
          t\n', 'It means that almost every operation has a module !\n']
```

```
In [123]: f = open('test.txt', 'rb')
          data3 = f.read()         # reads entire file as a single string
          f.close()
          print type(data3), "\n", data3

          <type 'str'>
          Python programming is interesting
          It is coming with batteries, in built
          It means that almost every operation has a module !
```

Interview Question : what is the advantage of performing file operations with context manager

In [124]:
```
with open('test.txt', 'rb') as f: # file operation with context manager
    data4 = f.read()
    f.close()

print type(data4), data4
```

```
<type 'str'> Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
```

Interview Question : How to add content add the end of file (EOF) in a file?

In [126]:
```
with open('test.txt', 'ab') as f:
    f.write("This is some line")
    f.close()

with open('test.txt', 'ab') as f:
    f.write("\n This is another line")
    # Observe the importance of '\n' escape sequence character, here.
    f.close()

with open('test.txt', 'rb+') as g:
    data5 = g.read()
    g.close()

print type(data5), '\n', data5
```

```
<type 'str'>
Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
This is some line
 This is another lineThis is some line
 This is another line
```

In [127]:
```
with open('test.txt', 'wb') as f:
    f.write("\n This is another line")
    f.close()

with open('test.txt', 'rb+') as g:
    data6 = g.read()
    g.close()

print type(data6), "\n", data6
# Observe that existing data is deleted, as the file is opened in write mode
```

```
<type 'str'>

 This is another line
```

In [128]:

```
#!/usr/bin/python
# fileOperations.py
'''
Purpose: File operations demonstration
'''

# accessing an exising file
f = open('test.txt', 'rb')
# 'f' is the file handler
print f, type(f)


# reading the data present in the test.txt
data = f.read()
print "data  = ", data

print 'Again trying to print the data from file'
data = f.read()
print "data  = ", data

print 'The current position of cursor in file is ', f.tell()

print 'moving the cursor position to 0'
f.seek(0)
print 'The current position of cursor in file is ', f.tell()

print 'The first 12 characters in the file are ', f.read(12)

print 'The next 6 characters in the file are \n', f.read(6)

print 'The current line in file is \n', f.readline()

print 'The current line in file is \n', f.readline()

print 'checking whether file is closed or not'
print 'return True, if the file is closed', f.closed

f.close()

print 'checking whether file is closed or not'
print 'return True, if the file is closed', f.closed

try:
    f.read() #No operation can be performed on a closed file object, as the ob
ject gets dereferenced
    # garbage collector deletes all the unreferenced objects
except ValueError, ve:
    print ve
    print "IO operation can't be performed on closed file handler"


g = open('test.txt', 'wb')   # opening existing file in read mode will erase i
ts existing data.

try:
    datag = g.read()
    print "datag = ", datag
```

```
except IOError, ex:
    print ex
    print 'opened file in write mode. can not read the data'

g.write('Python programming is interesting\n')
g.write('It is coming with batteries, in built\n')
g.write('It means that almost every operation has a module !')

g.close()  # it is not mandatory , but extremely recommended.
# python interpreter with close the file, but
# IronPython, Jython, ... may not close the file automatically.

# Using Context manager for file handling

with open('test.txt', 'ab+') as f:
    print 'The cursor position is at %d'%(f.tell())
    dataf = f.read()
    print 'The cursor position is at %d' % (f.tell())
    print 'The content of the data is \n', dataf
    # append mode supports both read and write operations
    print 'The cursor position is at %d' % (f.tell())
    print f.read(123)
    f.write('This is last line')
    print 'The cursor position is at %d' % (f.tell())
    f.close()
```

```
<open file 'test.txt', mode 'rb' at 0x038AF758> <type 'file'>
data  =
 This is another line
Again trying to print the data from file
data  =
The current position of cursor in file is  22
moving the cursor position to 0
The current position of cursor in file is  0
The first 12 characters in the file are
 This is an
The next 6 characters in the file are
other
The current line in file is
line
The current line in file is

checking whether file is closed or not
return True, if the file is closed False
checking whether file is closed or not
return True, if the file is closed True
I/O operation on closed file
IO operation can't be performed on closed file handler
File not open for reading
opened file in write mode. can not read the data
The cursor position is at 0
The cursor position is at 123
The content of the data is
Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
The cursor position is at 123

The cursor position is at 140
```

Assignment : Try to open the same file in two different modes, in parallel, and observe the problem occurring

Assignment : Write a script for camel casing to underscore casing, and vice versa; need to be done on an existing .py file. Ensure that functions imported from modules, are not disturbed.

Working with CSV files

Taking a sample csv file

```
sampleCSVFile.csv
    fruits, vegetables, cars
    Apple, Cabbagge, Benz
    Mango, Cucumber, Volvo
    Banana, Raddish, Maruthi suzuki
```

Creating a csv file

In [129]:
```
with open('sampleCSVFile.csv', 'ab+') as myCsv:
    myCsv.write("fruits, vegetables, cars\n")
    myCsv.write("Apple, Cabbagge, Benz\n")
    myCsv.write("Mango, Cucumber, Volvo\n")
    myCsv.write("Banana, Raddish, Maruthi suzuki\n")
    myCsv.close()
```

In [130]:
```
import os;
print os.listdir(os.getcwd())
```

```
['.ipynb_checkpoints', 'desktop.ini', 'fibGenerator.py', 'fibGenerator.pyc',
 'fileName.py', 'fileName.pyc', 'first', 'myfile.pyw', 'myNewFile.tsf', 'myNe
wFolder', 'newFolder', 'newScript.py', 'newScript.pyc', 'python - Basic Level
 Module.ipynb', 'python+-+Basic+Level+Module.html', 'sampleCSVFile.csv', 'tes
t.txt']
```

In [131]:
```
with open('sampleCSVFile.csv', 'rb') as c:
    data = c.read()
    c.close()

print type(data), '\n', data
```

```
<type 'str'>
fruits, vegetables, cars
Apple, Cabbagge, Benz
Mango, Cucumber, Volvo
Banana, Raddish, Maruthi suzuki
```

In [132]:
```
import csv
```

In [133]:
```
print dir(csv)
```

```
['Dialect', 'DictReader', 'DictWriter', 'Error', 'QUOTE_ALL', 'QUOTE_MINIMA
L', 'QUOTE_NONE', 'QUOTE_NONNUMERIC', 'Sniffer', 'StringIO', '_Dialect', '__a
ll__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__v
ersion__', 'excel', 'excel_tab', 'field_size_limit', 'get_dialect', 'list_dia
lects', 're', 'reader', 'reduce', 'register_dialect', 'unregister_dialect',
 'writer']
```

In [134]:
```
#!/usr/bin/python
# workingWithCSV.py
import csv
'''
    Purpose : Working with CSV files

'''
# with is called as a context manager
with open('sampleCSVFile.csv') as csvFile:
    data = csv.reader(csvFile, delimiter = ',')
    print data  # it is an iterator object
    for row in data:
        #print row
        print row[0]

    csvFile.close()
```

```
<_csv.reader object at 0x038797F0>
fruits
Apple
Mango
Banana
```

```
In [135]: with open('sampleCSVFile.csv') as csvFile:
              readCSV = csv.reader(csvFile, delimiter=',')    # delimiter
              print readCSV
              print type(readCSV)

              for row in readCSV:
                  print "row --> ", row
                  print "row[0] --> ",row[0]
                  print "row[0], row[1] --> ",row[0], row[1]
                  print '-'*70

              csvFile.close()
              # It is recommended to close the file, after it's use.
              # It deletes that file object.
              # so, file handler can't be used, after its closure.
```

```
<_csv.reader object at 0x038799F0>
<type '_csv.reader'>
row -->  ['fruits', ' vegetables', ' cars']
row[0] -->  fruits
row[0], row[1] -->  fruits  vegetables
----------------------------------------------------------------------
row -->  ['Apple', ' Cabbagge', ' Benz']
row[0] -->  Apple
row[0], row[1] -->  Apple  Cabbagge
----------------------------------------------------------------------
row -->  ['Mango', ' Cucumber', ' Volvo']
row[0] -->  Mango
row[0], row[1] -->  Mango  Cucumber
----------------------------------------------------------------------
row -->  ['Banana', ' Raddish', ' Maruthi suzuki']
row[0] -->  Banana
row[0], row[1] -->  Banana  Raddish
----------------------------------------------------------------------
```

Assignment : write a function to display all the cars, in the sampleCSVFile.csv

In [136]:
```
with open('sampleCSVFile.csv') as csvFile1:
    data = csv.reader(csvFile1, delimiter=',')
    print '\n',data

    for row in data:
        #cars = row[2]
        (fruits, vegetables, cars) = row    # tuple- unpacking
        print fruits, vegetables, cars

    csvFile1.close()
    print "outside the loop"
    print fruits, vegetables, cars
```

```
<_csv.reader object at 0x03879AB0>
fruits  vegetables  cars
Apple  Cabbagge  Benz
Mango  Cucumber  Volvo
Banana  Raddish  Maruthi suzuki
outside the loop
Banana  Raddish  Maruthi suzuki
```

In [137]:
```
with open('sampleCSVFile.csv') as csvFile1:
    data = csv.reader(csvFile1, delimiter=',')

    print '\n',data

    #vegetables = fruits = cars = []
    #It will result in all these objects will point to the same location
    vegetables = []
    fruits = []
    cars = []

    for index, row in enumerate(data):
        if index == 0:
            continue                 # to skip first iteration
        fruits.insert(index,row[0])
        #print fruits
        vegetables.insert(index,row[1])
        cars.insert(index,row[2])

    print '-'*50
    print 'fruits :',fruits
    print 'vegetables', vegetables
    print 'cars : ', cars

    print id(fruits), id(vegetables), id(cars)
    csvFile1.close()
```

```
<_csv.reader object at 0x03879B30>
--------------------------------------------------
fruits : ['Apple', 'Mango', 'Banana']
vegetables [' Cabbagge', ' Cucumber', ' Raddish']
cars :  [' Benz', ' Volvo', ' Maruthi suzuki']
58103848 59318800 59321080
```

In [138]:
```python
import csv

with open('names.csv', 'w') as csvfile:
    fieldnames = ['first_name', 'last_name']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerow({'first_name': 'Baked', 'last_name': 'Beans'})
    writer.writerow({'first_name': 'Lovely', 'last_name': 'Spam'})
    writer.writerow({'first_name': 'Wonderful', 'last_name': 'Spam'})
```

In [139]:
```python
import csv

inputNames = csv.DictReader(open("names.csv"))

for name in inputNames:
    print name
```

```
{'first_name': 'Baked', 'last_name': 'Beans'}
{'first_name': 'Lovely', 'last_name': 'Spam'}
{'first_name': 'Wonderful', 'last_name': 'Spam'}
```

Working with XML

In [1360]:
```python
from lxml import etree

# create XML
root = etree.Element('root')
root.append(etree.Element('child'))

# another child with text
child = etree.Element('child')
child.text = 'some text'
root.append(child)

# pretty string
s = etree.tostring(root, pretty_print=True)
print s
```

```
<root>
  <child/>
  <child>some text</child>
</root>
```

```
In [1361]: from xml.etree.ElementTree import Element, SubElement, tostring

           root = Element('root')
           child = SubElement(root, "child")
           child.text = "I am a child"

           print tostring(root)

           <root><child>I am a child</child></root>
```

Assignment : work with xml and xls files

Ref 1: http://xlsxwriter.readthedocs.io/worksheet.html#insert_image (http://xlsxwriter.readthedocs.io/worksheet.html#insert_image)

Ref 2: http://www.programcreek.com/python/example/56471/pygooglechart.PieChart2D (http://www.programcreek.com/python/example/56471/pygooglechart.PieChart2D)

Byte array

```
In [1362]: sentenceA = "Tomarrow is ours!!!"

           print type(sentenceA)

           <type 'str'>
```

```
In [1363]: sentenceB = bytearray("Tomarrow is ours!!!")

           print type(sentenceB)

           <type 'bytearray'>
```

NOTE: string objects are immutable, whereas bytearray objects are mutable

```
In [1364]: sentenceA[9:11]

Out[1364]: 'is'
```

```
In [1365]: sentenceA[9:11] = "will be"

           ---------------------------------------------------------------------------
           TypeError                                 Traceback (most recent call last)
           <ipython-input-1365-e72736eb4fc1> in <module>()
           ----> 1 sentenceA[9:11] = "will be"

           TypeError: 'str' object does not support item assignment
```

```
In [1366]: sentenceB[9:11]

Out[1366]: bytearray(b'is')
```

```
In [1367]: sentenceB[9:11] = "will be"
```

```
In [1368]: print sentenceB
```

```
Tomarrow will be ours!!!
```

```
In [1369]: print sentenceB[1], sentenceA[1]
```

```
111 o
```

```
In [1370]: chr(111)
```

```
Out[1370]: 'o'
```

```
In [1371]: ord('o')
```

```
Out[1371]: 111
```

NOTE: bytearray will store the characters with their corresponding ASCII values

Data Serialization

```
- converting the objects into byte form .
- Used for transferring the objects.
- Serialization is the process of converting a data structure or object state into
 a format that can be stored.
- Serialization is also called deflating or marshalling
- DeSerialization is also called Inflating or unmarshalling.
```

Various ways of data serialization

```
- Marshall -- It is primitive, and no more used.
- Pickle   -- Pickle is a standard module which serializes and deserializes a pytho
n object structure.
- cPickle  --  c implementation of pickle]
- shelve   -- Advanced version, to address the security flaws of pickel/cpickle
- xml
- json     -- Most popular, now-a-days
- db
```

Python 2.x has both pickle and cpickle. Whereas python 3.x has only cPickle, and it is renamed as pickle.

Pickle files has .pkl or .pickle extensions. The pickled data format is python specific.

Interview Question : Difference between compression and serialization ?

Ans: compression may be lossy or lossless process. Whereas Serialization is a lossless reversible process. compression is used to reduce the data redundancy, whereas serialization is used for inflating or deflating an object, and communicating data with other languages.

In [1373]:
```python
#!/usr/bin/python
# workingWithPickle.py

import pickle

'''
    Purpose: Working with Pickle files
    pickling

'''

# Serialization
students = ['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 'Naseer','Shoban', 123]

f = open('Students.pickle', 'ab+')
pickle.dump(students, f)
f.flush()

f.close()

# Deserialization
g = open('Students.pickle', 'rb')
myStudents = pickle.load(g)
print "myStudents are ", myStudents
g.close()
```

```
myStudents are  ['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 'Naseer', 'Shoban',
 123]
```

Interview Question : what is the difference between dump and dumps methods

In [1374]:
```
# serialization
mystud = pickle.dumps(students)    # dumping into a string
print mystud
print type(mystud)
```

```
(lp0
S'Mujeeb'
p1
aS'Harini'
p2
aI01
aS'Ankit'
p3
aS'Naseer'
p4
aS'Shoban'
p5
aI123
a.
<type 'str'>
```

In [162]:
```
# Deserialization
orgStud = pickle.loads(mystud)        # Loading from a string
print orgStud
print type(orgStud)

print orgStud == students
```

```
['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 'Naseer', 'Shoban', 123]
<type 'list'>
True
```

In [163]:
```
with open('Students.pickle', 'rb+') as ktf:
    myStudents = pickle.load(ktf)
print "myStudents are ", myStudents
```

```
myStudents are  ['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 'Naseer', 'Shoban',
 123]
```

In **conclusion**, Pickle and cpickle has their importance in interfacing with c and C++. As pickled data format is python specific, it is not used in interfacing with other languages.

In [164]:
```
def func():
    pass

z = func()

print z, type(z)
print func, type(func)
```

```
None <type 'NoneType'>
<function func at 0x03912730> <type 'function'>
```

```
In [165]: try:
              zz = pickle.dumps(z)
          except pickle.PickleError as pe:
              print 'The error is ', pe
```

```
In [166]: print zz

          N.
```

Interview Question : When a function object is deleted, but the object created with that function call is retained, will it malfunction, or work properly?

```
In [167]: del func

          print zz

          N.
```

```
In [168]: try:
              zz1 = pickle.dumps(z)
          except pickle.PickleError as pe:
              print 'The error is ', pe
```

```
In [169]: print zz1

          N.
```

```
In [170]: class class1():
              pass

          b = class1()

          pickle.dumps(b)
          del class1
          print b

          <__main__.class1 instance at 0x038FF350>
```

Though class is deleted, its instance survivies; same as the case with functions

```
In [171]: pickle.dumps('I am pickling')

Out[171]: "S'I am pickling'\np0\n."
```

```
In [172]: pickle.loads("S'I am pickling'\np0\n.")

Out[172]: 'I am pickling'
```

```
In [173]: pickle.loads(pickle.dumps('I am pickling'))

Out[173]: 'I am pickling'
```

Limitations of Pickle

- The pickle module is not secure against erroneous or maliciously constructed data.
- The pickled data can be modfied on-the-fly, mainly by Middle-man attack.
- Never unpickle data received from an untrusted or unauthenticated source.

Shelve

- shelve is a tool that uses pickle to store python objects in an access-by-key file system. It is same as keys in dictionary.
- It is used as a simple persistent storage option for python objects when a relational database is overkill.
- The values are pickled and written to a database created and managed by anydbm.
- anydbm is a frontend interface to establish communication with database.

In [174]:
```
import shelve
```

In [175]:
```
s= shelve.open('usingShelve.db')

try:
    s['key1'] = {'int': 8, 'float': 8.0, 'string': '8'}
except Exception, ex:
    print ex
finally:
    s.close()
```

In [176]:
```
# To access the data again,

s = shelve.open('usingShelve.db')  # default is read only mode
try:
    myShelveContent = s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()

print 'myShelveContent = ', myShelveContent
```

```
myShelveContent =  {'int': 8, 'float': 8.0, 'string': '8'}
```

opening shelve in read-only mode.

In [177]:
```
s = shelve.open('usingShelve.db', flag = 'r')
try:
    myShelveContent = s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()

print 'myShelveContent = ', myShelveContent
```

```
myShelveContent =  {'int': 8, 'float': 8.0, 'string': '8'}
```

In [178]:
```
s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
    s['key1']['newValue'] = 'This is a new value'
    # trying to write data in read mode; observe that this line has no reflect
ion
    print s['key1']
except Exception, ex1:
    print ex1
finally:
    s.close()

s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()
```

```
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8'}
```

NOTE: By defaults, shelve do not track modifications to volatile objects. If necessary, open the shelve file in writeback enabled

In [179]:
```
s = shelve.open('usingShelve.db', writeback=True)
try:
    print s['key1']   # accessing using the key
    s['key1']['newValue'] = 'This is a new value'
    print s['key1']
except Exception, ex1:
    print ex1
finally:
    s.close()

s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()
```

```
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8', 'newValue': 'This is a new value'}
{'int': 8, 'float': 8.0, 'string': '8', 'newValue': 'This is a new value'}
```

Note: Shelve must not be opened with writeback enabled, unless it is essential

JSON

- JSON is an abbreviation for Java Script Object notation
- json is supported by almost all languages.
- official website is json.org
- stored in .json file. But, it can be stored in other format too, like .template file in AWS cloudformation, etc.

sample json

```
{
    "employees": {
        "employee": [
            {
                "id": "1",
                "firstName": "Tom",
                "lastName": "Cruise",
                "photo": "http://cdn2.gossipcenter.com/sites/default/files/imagecac
he/story_header/photos/tom-cruise-020514sp.jpg"
            },
            {
                "id": "2",
                "firstName": "Maria",
                "lastName": "Sharapova",
                "photo": "http://thewallmachine.com/files/1363603040.jpg"
            },
            {
                "id": "3",
                "firstName": "James",
                "lastName": "Bond",
                "photo": "http://georgesjournal.files.wordpress.com/2012/02/007_at_
50_ge_pierece_brosnan.jpg"
            }
        ]
    }
}
```

In [180]:
```
import json
```

In [181]:
```
students = {'batch 1' : {'Name': 'Ankit', 'regNumber': 1},
             'batch 2' : {'Name': 'Mujeeb', 'regNumber': 2}
            }
```

In [182]:
```
print json.dumps(students)  # dictionary to json
```

```
{"batch 2": {"regNumber": 2, "Name": "Mujeeb"}, "batch 1": {"regNumber": 1,
 "Name": "Ankit"}}
```

In [183]:
```
print json.dumps(students, sort_keys = True)
```

```
{"batch 1": {"Name": "Ankit", "regNumber": 1}, "batch 2": {"Name": "Mujeeb",
 "regNumber": 2}}
```

tuple to json array

```
In [184]: tup1 = ('Red', 'Black', 'white', True, None, [])
          json.dumps(tup1)

Out[184]: '["Red", "Black", "white", true, null, []]'
```

Observe that True is changed to true; and None to null

```
In [185]: string = 'This is a string'
          json.dumps(string)

Out[185]: '"This is a string"'

In [186]: b = True
          json.dumps(b)

Out[186]: 'true'

In [187]: a = -123
          b = -2.34
          z = 1.3e-10

In [188]: json.dumps(a)

Out[188]: '-123'

In [189]: json.dumps(b)

Out[189]: '-2.34'

In [190]: json.dumps(z)

Out[190]: '1.3e-10'
```

Interview Question : How different is python dictionary, from that of json?

json is different from dictionary.

```
- All the content within the dictionary must be in key-value pairs; whereas there c
an be arrays in dictionary.
- And, json data types are different from that of python data types.
```

json doesn't cover all the datatypes of python, mainly tuples and bytes.

Decoding the json data, back to python data types can be achieved using json.loads correspondingly.

Assignment : Use json.loads to correspondingly deserialize it

json to python object conversion pairs:

JSON	Python
object	dict
array	list
string	str
number(int)	int
number(real)	float
true	True
false	False
null	None

There are various online json lints to validate a written json file; and online json to other format convertors, like code Beautify (http://codebeautify.org/jsonviewer/), json formatter (https://jsonformatter.curiousconcept.com/), etc

In [1375]:

```python
import json

input = '''
[
  { "id" : "001",
    "x" : "2",
    "name" : "Chuck"
  } ,
  { "id" : "009",
    "x" : "7",
    "name" : "Chuck"
  }
]'''

info = json.loads(input)
print('User count:', len(info))

for item in info:
    print('Name', item['name'])
    print('Id', item['id'])
    print('Attribute', item['x'])
```

```
('User count:', 2)
('Name', u'Chuck')
('Id', u'001')
('Attribute', u'2')
('Name', u'Chuck')
('Id', u'009')
('Attribute', u'7')
```

Working with XML

In [1376]:
```python
import xml.etree.ElementTree as ET

data = '''
<person>
  <name>Rama Rao</name>
  <phone type="intl">
     +91 7878787878
   </phone>
   <email hide="yes"/>
</person>'''

tree = ET.fromstring(data)
print('Name:',tree.find('name').text)
print('Attr:',tree.find('email').get('hide'))
```
```
('Name:', 'Rama Rao')
('Attr:', 'yes')
```

In [1377]:
```python
import xml.etree.ElementTree as ET
'''
XML - Looping through nodes
'''
input = '''
<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
            </user>
        </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```
```
('User count:', 2)
('Name', 'Chuck')
('Id', '001')
('Attribute', '2')
('Name', 'Brent')
('Id', '009')
('Attribute', '7')
```

Web services

Assignment : Go through this Yahoo weblink (https://developer.yahoo.com/python/python-rest.html) and have an insight on accessing web content using python.

In [191]:
```
import urllib

url = 'https://developer.yahoo.com/'
u = urllib.urlopen(url)
# u is a file-like object
data = u.read()
f = open('yahooData.txt', 'ab+')

f.write(data)
f.close()
```

NOTE: Both urllib and urllib serve the same purpose; But, urllib2 module can handle HTTP Errors better than urllib

In [192]:
```
import urllib2

try:
        data = urllib2.urlopen(url).read()
except urllib2.HTTPError, e:
        print "HTTP error: %d" % e.code
except urllib2.URLError, e:
        print "Network error: %s" % e.reason.args[1]

with open('yahooData1.txt', 'ab+') as g:
    g.write(data)
    g.close()
```

webservices: urllib, urllib2, BeautifulSoup (bs4), httplib, cookielib

Practical Example: Working with googlde geocoding API

In [1378]:
```
import urllib
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

while True:
    address = raw_input('Enter location: ')  # Hyderabad, IN
    if len(address) < 1 : break

    url = serviceurl + urllib.urlencode(
        {'sensor':'false', 'address': address})

    print('Retrieving', url)
    uh = urllib.urlopen(url)
    data = uh.read().decode()
    print('Retrieved',len(data),'characters')

    try:
        js = json.loads(data)
    except:
        js = None

    if not js or 'status' not in js or js['status'] != 'OK':
        print('==== Failure To Retrieve ====')
        print(data)
        continue

    print(json.dumps(js, indent=4))

    lat = js["results"][0]["geometry"]["location"]["lat"]
    lng = js["results"][0]["geometry"]["location"]["lng"]
    print('lat',lat,'lng',lng)
    location = js['results'][0]['formatted_address']
    print(location)
    print '-'*80
    choice = raw_input("Do you want to retry: Y or N: ")
    if choice.lower() == 'n': break
```

```
Enter location: Hyderabad, India
('Retrieving', 'http://maps.googleapis.com/maps/api/geocode/json?sensor=false
&address=Hyderabad%2C+India')
('Retrieved', 1542, 'characters')
{
    "status": "OK",
    "results": [
        {
            "geometry": {
                "location_type": "APPROXIMATE",
                "bounds": {
                    "northeast": {
                        "lat": 17.6078088,
                        "lng": 78.6561694
                    },
                    "southwest": {
                        "lat": 17.2168886,
                        "lng": 78.1599217
                    }
                },
                "viewport": {
                    "northeast": {
                        "lat": 17.6078088,
                        "lng": 78.6561694
                    },
                    "southwest": {
                        "lat": 17.2168886,
                        "lng": 78.1599217
                    }
                },
                "location": {
                    "lat": 17.385044,
                    "lng": 78.486671
                }
            },
            "address_components": [
                {
                    "long_name": "Hyderabad",
                    "types": [
                        "locality",
                        "political"
                    ],
                    "short_name": "Hyderabad"
                },
                {
                    "long_name": "Telangana",
                    "types": [
                        "administrative_area_level_1",
                        "political"
                    ],
                    "short_name": "Telangana"
                },
                {
                    "long_name": "India",
                    "types": [
                        "country",
                        "political"
                    ],
                    "short_name": "IN"
                }
            ],
            "place_id": "ChIJx9Lr6tqZyzsRwvu6koO3k64",
            "formatted_address": "Hyderabad, Telangana, India",
            "types": [
                "locality",
                "political"
            ]
        }
    ]
}
('lat', 17.385044, 'lng', 78.486671)
Hyderabad, Telangana, India
-----------------------------------------------------------------------------
---
Do you want to retry: Y or N: N
```

```
In [1379]: import urllib
           import xml.etree.ElementTree as ET

           serviceurl = 'http://maps.googleapis.com/maps/api/geocode/xml?'

           while True:
               address = raw_input('Enter location: ')
               if len(address) < 1: break

               url = serviceurl + urllib.urlencode({'address': address})
               print('Retrieving', url)
               uh = urllib.urlopen(url)
               data = uh.read()
               print('Retrieved', len(data), 'characters')
               print(data.decode())
               tree = ET.fromstring(data)

               results = tree.findall('result')
               lat = results[0].find('geometry').find('location').find('lat').text
               lng = results[0].find('geometry').find('location').find('lng').text
               location = results[0].find('formatted_address').text

               print('lat', lat, 'lng', lng)
               print(location)
               print '-' * 80
               choice = raw_input("Do you want to retry: Y or N: ")
               if choice.lower() == 'n': break
```

```
Enter location: Hyderabad, India
('Retrieving', 'http://maps.googleapis.com/maps/api/geocode/xml?address=Hyder
abad%2C+India')
('Retrieved', 1360, 'characters')
<?xml version="1.0" encoding="UTF-8"?>
<GeocodeResponse>
 <status>OK</status>
 <result>
  <type>locality</type>
  <type>political</type>
  <formatted_address>Hyderabad, Telangana, India</formatted_address>
  <address_component>
   <long_name>Hyderabad</long_name>
   <short_name>Hyderabad</short_name>
   <type>locality</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>Telangana</long_name>
   <short_name>Telangana</short_name>
   <type>administrative_area_level_1</type>
   <type>political</type>
  </address_component>
  <address_component>
   <long_name>India</long_name>
   <short_name>IN</short_name>
   <type>country</type>
   <type>political</type>
  </address_component>
  <geometry>
   <location>
    <lat>17.3850440</lat>
    <lng>78.4866710</lng>
   </location>
   <location_type>APPROXIMATE</location_type>
   <viewport>
    <southwest>
     <lat>17.2168886</lat>
     <lng>78.1599217</lng>
    </southwest>
    <northeast>
     <lat>17.6078088</lat>
     <lng>78.6561694</lng>
    </northeast>
   </viewport>
   <bounds>
    <southwest>
     <lat>17.2168886</lat>
     <lng>78.1599217</lng>
    </southwest>
    <northeast>
     <lat>17.6078088</lat>
     <lng>78.6561694</lng>
    </northeast>
   </bounds>
  </geometry>
  <place_id>ChIJx9Lr6tqZyzsRwvu6koO3k64</place_id>
 </result>
</GeocodeResponse>

('lat', '17.3850440', 'lng', '78.4866710')
Hyderabad, Telangana, India
-------------------------------------------------------------------------------
---
Do you want to retry: Y or N: N
```

Debugger in Python

Debugging is essential to understand the call flow in run time. It is also used to debug an error.

There are various debuggers available in python, as listed here (https://wiki.python.org/moin/PythonDebuggingTools). These debuggers can be used in

1. Interactive mode, or
2. within the IDE

In all the cases, it is not feasible to work with an IDE. ex: Devops jobs of logging into a remote linux/Windows server, to solve an issue. Especially, in such cases, interactive debuggers such as pdb, ipdb, ... can be helpful.

pydev

- Majority of the IDEs will have pydev configured in it.
- Basic features of pydev are
  - step Over
  - step In
  - step Out
  - Placing debugger
  - Visualizing the objects in debug mode ....

pdb

- It is the basic interactive Debugger
- Features
  - pause a program
  - look at the values of variables
  - watch program execution step-by-step

Usage:

1. In the terminal :

   $ python -m pdb fileName.py
2. Within the script using pdb module

   import pdb

In the script, place the following statement from where we want to make debugging.

```
pdb.set_trace()
```

Now, run the program normally

At the output console, it returns pdb prompt, and wait for the commands.

```
(pdb)  l          # To list the complete script, irrespective of the line in which
 the set_trace() statement is placed.
(pdb)  l 1,5     # To list 1st line to 5th line (including) irrespective of place i
n which set_trace() statement is placed.
(pdb)  c        # Continue execution till a break point is encountered




(pdb) help        # To get help find all the options possible with pdb
(pdb) n         # Continue execution until the next line in current function is rea
ched or it returns
(pdb) r         # continue execution, until the current function returns
(pdb) u         # shows the flow. Shows previous step in execution flow. but, it wi
ll not execute
(pdb) d         # shows next step in execution flow.
```

Most commonly used among them are:

```
l(ist)
n(ext)
c(ontinue)
s(tep)
r(eturn)
b(reak)
```

```
    pdb ? - to get help

    pdb l - show the cursor position

    pdb l 18 - to list line 18 in file
    pdb passesLeft  - to get the number of passes left

    pdb <any variable> - to get the value in variable

    pdb b 18 - to place a breakpoint
    pdb l

    pdb n - to execute next step
```

pdb comes with four modes of operation:

```
    - Script, postmortem, run and Trace modes
```

Working with pdb, within the python scripts

save the below scripts, and execute them in terminal, like general python scripts.

In [193]:
```python
#!/usr/bin/python
# first.py
import pdb
version = '3.0'
def hello():
  ''' this is just for printing hello '''
  print "hello today is modules class"
def add(a=1,b=2):
  ''' this is an addition program '''
  sum = a + b
  return sum


pdb.set_trace()
if __name__ == '__main__':
  hello()
  sum = add()
  sum1 = add(5,6)
  print sum1
  print sum
  print version
else:
  print "Hello this has to be part of my modules"
```

```
--Return--
> <ipython-input-193-ca0a9d73001e>(14)<module>()->None
-> pdb.set_trace()
(Pdb) l
  9       ''' this is an addition program '''
 10       sum = a + b
 11       return sum
 12
 13
 14  -> pdb.set_trace()
 15     if __name__ == '__main__':
 16       hello()
 17       sum = add()
 18       sum1 = add(5,6)
 19       print sum1
(Pdb) l 9,15
  9       ''' this is an addition program '''
 10       sum = a + b
 11       return sum
 12
 13
 14  -> pdb.set_trace()
 15     if __name__ == '__main__':
(Pdb) n
> c:\python27\lib\site-packages\ipython\core\interactiveshell.py(2884)run_cod
e()
-> sys.excepthook = old_excepthook
(Pdb) c
hello today is modules class
11
3
3.0
```

In [194]:
```
#!/usr/bin/python
# second.py
import pdb
pdb.set_trace()
import first as f
print f.hello()
print f.add(10,20)
print f.version
```

```
--Return--
> <ipython-input-194-6c3a48b9f2b1>(4)<module>()->None
-> pdb.set_trace()
(Pdb) help

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv

(Pdb) ?

Documented commands (type help <topic>):
========================================
EOF    bt         cont      enable  jump  pp       run      unt
a      c          continue  exit    l     q        s        until
alias  cl         d         h       list  quit     step     up
args   clear      debug     help    n     r        tbreak   w
b      commands   disable   ignore  next  restart  u        whatis
break  condition  down      j       p     return   unalias  where

Miscellaneous help topics:
==========================
exec  pdb

Undocumented commands:
======================
retval  rv

(Pdb) b 6
Breakpoint 1 at <ipython-input-194-6c3a48b9f2b1>:6
(Pdb) l
  1     #!/usr/bin/python
  2     # second.py
  3     import pdb
  4  -> pdb.set_trace()
  5     import first as f
  6 B   print f.hello()
  7     print f.add(10,20)
  8     print f.version
[EOF]
(Pdb) b 7
Breakpoint 2 at <ipython-input-194-6c3a48b9f2b1>:7
(Pdb) c
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-194-6c3a48b9f2b1> in <module>()
      3 import pdb
      4 pdb.set_trace()
----> 5 import first as f
      6 print f.hello()
      7 print f.add(10,20)

ImportError: No module named first
```

Note: You can observe some code which is getting executed here, though it is not part of our second.py. It is beacuse of ipython.

Also, with this debugging, we clearly identified that 'first.py' is not located in the current working directory.

Now, after placing **init**.py file in 'debugging' folder, It got executed.

In [196]:
```python
#!/usr/bin/python
# second.py
import pdb
pdb.set_trace()
import debugging.first as f
print f.hello()
print f.add(10,20)
print f.version
```

```
--Return--
> <ipython-input-196-977263d8e280>(4)<module>()->None
-> pdb.set_trace()
(Pdb) c
> c:\users\home\google drive\python\tut\complete material\debugging\first.py
(15)<module>()
-> if __name__ == '__main__':
(Pdb) c
Hello this has to be part of my modules
hello today is modules class
None
30
3.0
```

In [1]:
```
#!/usr/bin/python
# third.py
import pdb

for i in (1,2,3):
  pdb.set_trace()
  print i
```

```
> <ipython-input-1-0bad0586cb8f>(7)<module>()
-> print i
(Pdb) n
1
> <ipython-input-1-0bad0586cb8f>(5)<module>()
-> for i in (1,2,3):
(Pdb) l
  1     #!/usr/bin/python
  2     # third.py
  3     import pdb
  4
  5  -> for i in (1,2,3):
  6       pdb.set_trace()
  7       print i
[EOF]
(Pdb) n
> <ipython-input-1-0bad0586cb8f>(6)<module>()
-> pdb.set_trace()
(Pdb) l
  1     #!/usr/bin/python
  2     # third.py
  3     import pdb
  4
  5     for i in (1,2,3):
  6  ->   pdb.set_trace()
  7       print i
[EOF]
(Pdb) s
--Call--
> c:\python27\lib\pdb.py(1250)set_trace()
-> def set_trace():
(Pdb) s
> c:\python27\lib\pdb.py(1251)set_trace()
-> Pdb().set_trace(sys._getframe().f_back)
(Pdb) s
--Call--
> c:\python27\lib\pdb.py(61)__init__()
-> def __init__(self, completekey='tab', stdin=None, stdout=None, skip=None):
(Pdb) r
--Return--
> c:\python27\lib\pdb.py(104)__init__()->None
-> self.commands_bnum = None # The breakpoint number for which we are
(Pdb) r
--Return--
> c:\python27\lib\pdb.py(1251)set_trace()->None
-> Pdb().set_trace(sys._getframe().f_back)
(Pdb) r
> <ipython-input-1-0bad0586cb8f>(7)<module>()
-> print i
(Pdb) c
2
> <ipython-input-1-0bad0586cb8f>(6)<module>()
-> pdb.set_trace()
(Pdb) c
3
```

Similarly, in fourth.py, you can observe the functionality of for loop and if condition, in runtime.

In [2]:
```
#!/usr/bin/python
# fourth.py
import pdb

for i in range(1,11):
  if i == 5:
    pdb.set_trace()
    continue
  print i
```

```
1
2
3
4
> <ipython-input-2-f1378680eab5>(8)<module>()
-> continue
(Pdb) l
  3     import pdb
  4
  5     for i in range(1,11):
  6       if i == 5:
  7         pdb.set_trace()
  8  ->     continue
  9       print i
[EOF]
(Pdb) b 5,6,9
Breakpoint 1 at <ipython-input-2-f1378680eab5>:5
(Pdb) l
[EOF]
(Pdb) n
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
(Pdb) l
  1     #!/usr/bin/python
  2     # fourth.py
  3     import pdb
  4
  5 B-> for i in range(1,11):
  6       if i == 5:
  7         pdb.set_trace()
  8         continue
  9       print i
[EOF]
(Pdb) d 5
*** Newest frame
(Pdb) l 1,9
  1     #!/usr/bin/python
  2     # fourth.py
  3     import pdb
  4
  5 B-> for i in range(1,11):
  6       if i == 5:
  7         pdb.set_trace()
  8         continue
  9       print i
(Pdb) c
6
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
(Pdb) c
7
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
(Pdb) c
8
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
```

```
(Pdb) r
9
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
(Pdb) r
10
> <ipython-input-2-f1378680eab5>(5)<module>()
-> for i in range(1,11):
(Pdb) c
```

Now, let try to debug a script with some bugs.

Assignment : Try to find the dug, in this buggyFifth.py script.

```
#!/usr/bin/python
# buggyFifth.py
def eo(num):
  ''' fun for even odd numbers '''
  if num % 2 == 0:
    return 'even'
  else:
    return 'odd'

import pdb;pdb.set_trace()
print eo(2)
print eo(3)
print eo.func_doc
```

Assignment : Analyze the script execution in this fifth.py

```
#!/usr/bin/python
# fifth.py
def eo(num):
  ''' fun for even odd numbers '''
  if num % 2 == 0:
    print 'even'
  else:
    print 'odd'

import pdb
pdb.set_trace()
for i in range(100):
  print "The number1 is %s" %(i)
  print "The number2 is %s" %(i)
  print "The number3 is %s" %(i)
  print "The number4 is %s" %(i)
  print "The number5 is %s" %(i)
  eo(i)
```

Assignment : In this sixth.py, Observe the call flow of function calls.

```
#!/usr/bin/python
# sixth.py
import pdb

def fourth():
  second()
  print "this is my fourth function \n"
def third():
  print "this is my third function \n"
def second():
  third()
  print "this is my second function \n"
def first():
  fourth()
  print "this is my first function \n"

pdb.set_trace()
first()
```

In conclusion, pdb debugging starts from the line where the pdb.set_trace() is placed. It can be more helpful, in cases where using IDE or other GUI based debuggers is not possible. But, there are other debuggers such as pydev, ... which have much more functionality than pdb.

re module

Regular expression (or regex) are used for pattern matching in the date. It is available in almost all programming languages. Python has re module to deal with regular expressions.

```
In [3]: import re
```

```
In [4]: print dir(re)
```

```
['DEBUG', 'DOTALL', 'I', 'IGNORECASE', 'L', 'LOCALE', 'M', 'MULTILINE', 'S',
 'Scanner', 'T', 'TEMPLATE', 'U', 'UNICODE', 'VERBOSE', 'X', '_MAXCACHE', '__
all__', '__builtins__', '__doc__', '__file__', '__name__', '__package__', '__
version__', '_alphanum', '_cache', '_cache_repl', '_compile', '_compile_rep
l', '_expand', '_locale', '_pattern_type', '_pickle', '_subx', 'compile', 'co
py_reg', 'error', 'escape', 'findall', 'finditer', 'match', 'purge', 'searc
h', 'split', 'sre_compile', 'sre_parse', 'sub', 'subn', 'sys', 'template']
```

```
In [5]: regObject = re.compile("python")     # creating a regular expression Object, r
        egObject
```

```
In [6]: print regObject, type(regObject)
```

```
<_sre.SRE_Pattern object at 0x038802E0> <type '_sre.SRE_Pattern'>
```

re.match()

re.match(string[, pos[, endpos]])

- Matches zero or more characters at the beginning of the string

```
In [7]: regObject.match("python Programming is Good")    # The matched result is store
        d in re type object
```

```
Out[7]: <_sre.SRE_Match at 0x38edb80>
```

```
In [8]: print regObject.match("python Programming is Good")
```

```
<_sre.SRE_Match object at 0x038EDF00>
```

```
In [9]: print regObject.match("python Programming is Good").group() # returns the matc
        hes stored in object
```

```
python
```

```
In [10]: print regObject.match("Programming python is Good")
```

```
None
```

```
In [11]: print regObject.match("Programming python is Good").group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-11-bba0226c0a4e> in <module>()
----> 1 print regObject.match("Programming python is Good").group()

AttributeError: 'NoneType' object has no attribute 'group'
```

Until and unless there are matches found, the group() attribute will not be present.

```
In [12]: print regObject.match(" python Programming is Good")  # white-space in startin
g
```

```
None
```

```
In [13]: print regObject.match("Python Programming is Good")    # capital 'P' encounter
ed
```

```
None
```

```
In [14]: print regObject.match('pythoN')  # results in None
```

```
None
```

```
In [15]: print regObject.match('pythoN'.lower())
```

```
<_sre.SRE_Match object at 0x0396A1E0>
```

```
In [16]: #!/usr/bin/python
import os

checkString = 'python'
compiledCheckString = re.compile(checkString)   # re object will be created
targetString = 'python programming'
matchs = compiledCheckString.match(targetString)    # match() will be initiate
d on the re object
print matchs
if matchs:
    print "Now lets print the matches word"
    print matchs.group()
```

```
<_sre.SRE_Match object at 0x0396A250>
Now lets print the matches word
python
```

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

         checkString = 'python'
         compiledCheckString = re.compile(checkString)   # re object will be created
         targetString = 'Python'
         matchs = compiledCheckString.match(targetString)    # match() will be initiate
         d on the  re object
         print matchs
         if matchs:
             print "Now lets print the matches word"
             print matchs.group()
         else:
             print "No matches found"

         None
         No matches found
```

Interview Question : What is the difference between re.search() and re.match()?

re.search()

re.search(string[, pos[, endpos]])

- Matches zero or more characters anywhere in the string

```
In [18]: regObject = re.compile("python")

In [19]: regObject.search("python programming")

Out[19]: <_sre.SRE_Match at 0x396a250>

In [20]: regObject.match("python programming")

Out[20]: <_sre.SRE_Match at 0x396a3d8>

In [21]: regObject.search("programming python is good")

Out[21]: <_sre.SRE_Match at 0x396a448>

In [22]: print regObject.match("programming python is good")  # match tries to match in
          the starting only

         None

In [23]: regObject.search("programming is good in python")

Out[23]: <_sre.SRE_Match at 0x396a528>
```

```
In [24]: regObject = re.compile("python is")

         print regObject.match("programming python is good")
         print regObject.search("programming python is good")

         None
         <_sre.SRE_Match object at 0x0396A1E0>
```

```
In [25]: print regObject.search("is programming python in good ?")

         None
```

Special Characters

^ (caret) - Matches the start of the string

```
In [26]: string = "This is python class"
         regObject = re.compile("^This")
```

```
In [27]: result = regObject.match(string)
```

```
In [28]: print result.group()

         This
```

Doing in other way

```
In [29]: re.match('^This',string)

Out[29]: <_sre.SRE_Match at 0x396a800>
```

```
In [30]: re.match('^This',string).group()

Out[30]: 'This'
```

```
In [31]: re.match('^This',"This is python class").group()

Out[31]: 'This'
```

```
In [32]: re.search('^This',"This is python class").group()

Out[32]: 'This'
```

```
In [33]: print re.search('^This', "Yes!, This is python class")

         None
```

$ - matches the end of the string, or just before the newline at the end of the string

```
In [34]: #!/usr/bin/python
         import re
         string = 'foo foobar'
         print string
         regObject= re.compile('foobar$')
         print "regObject = ",regObject
         print "regObject.match(string) ", regObject.match(string)    # re.match() vs r
         e.search()
         print "regObject.search(string) ", regObject.search(string)
```

```
foo foobar
regObject =  <_sre.SRE_Pattern object at 0x038245E0>
regObject.match(string)  None
regObject.search(string)  <_sre.SRE_Match object at 0x0396A8A8>
```

```
In [35]: print re.match('foobar$',string)  # pattern match only in the starting of line
```

```
None
```

```
In [36]: print re.search('foobar$',string)
```

```
<_sre.SRE_Match object at 0x0396A9C0>
```

```
In [37]: print re.search('foobar$',string).group()
```

```
foobar
```

. (DOT)

- matches any character, except the newline. if DOTALL flag is enabled, it matches any character including newline

```
In [38]: #!usr/bin/python

         string = 'This'
         result = re.search('....', string)

         if result:
             print result.group()
```

```
This
```

```
In [39]: print re.search('......', "Shoban").group()
```

```
Shoban
```

```
In [40]: print re.search('....', "Shoban").group()
```

```
Shob
```

```
In [41]: print re.search('', "Shoban").group()
In [42]: print re.search('....$', "Shoban").group()

         oban
```

```
In [43]: print re.search('^....', "Shoban").group()

         Shob
```

```
In [44]: print re.search('^....$', "Shoban").group()

         ---------------------------------------------------------------------------
         AttributeError                            Traceback (most recent call last)
         <ipython-input-44-42e3a946b9a3> in <module>()
         ----> 1 print re.search('^....$', "Shoban").group()

         AttributeError: 'NoneType' object has no attribute 'group'
```

```
In [45]: print re.search('^......$', "Shoban").group()

         Shoban
```

* causes the RE to match 0 or more repetitions of the preceeding RE.

```
ex: 'ab*'  - matches for 'a', 'ab', and 'a' followed by any number of 'b''s
                    ab, abb, abbbb ...
```

```
In [46]: print string

         This
```

```
In [47]: re.search('.*',string).group()

Out[47]: 'This'
```

```
In [48]: re.search('.*', "I am aspiring to be a promising Engineer").group()

Out[48]: 'I am aspiring to be a promising Engineer'
```

```
In [49]: re.search('.a*', "I am aspiring to be a promising Engineer").group()

Out[49]: 'I'
```

```
In [50]: re.search('a*', "I am aspiring to be a promising Engineer").group()

Out[50]: ''
```

```
In [51]: re.search('. *', "I am aspiring to be a promising Engineer").group()

Out[51]: 'I '
```

In [52]:
```
re.search('.g*', "I am aspiring to be a promising Engineer").group()   #'g' is
 occuring 0 times
```
Out[52]:
```
'I'
```

In [55]:
```
re.search('.*', '').group()   # trying to find in null string   # resulted in
 white-space, not NONE
```
Out[55]:
```
''
```

In [56]:
```
re.search('', '').group()   # trying to find in null string     # resulted in
 white-space, not NONE
```
Out[56]:
```
''
```

+

- causes the RE to match 1 or more repetitions of the preceeding RE

ex: ab+ -> 'ab', 'abb', 'abbb', ...

In [57]:
```
string = 'abbba'

re.match('ab+a',string).group()
```
Out[57]:
```
'abbba'
```

In [58]:
```
print re.match('ab+a','aa')
```
```
None
```

In [59]:
```
print re.match('ab*a','aa123').group()     # * tries to find 0 or more occurre
nces of 'b'
```
```
aa
```

In [60]:
```
print re.match('ab*c','aa123').group()
```
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-60-b324747346bc> in <module>()
----> 1 print re.match('ab*c','aa123').group()

AttributeError: 'NoneType' object has no attribute 'group'
```

In [61]:
```
print re.match('ab*c','aca123').group()
```
```
ac
```

In [62]:
```
print re.match('ab*.*','aca123').group()
```
```
aca123
```

```
In [63]: re.search('ab+a','abbba').group()

Out[63]: 'abbba'
```

```
In [64]: print re.search('ab+a','aa')

None
```

? causes the RE to match 0 or 1 time of the preceeding RE

ex: ab? --> 'a', 'ab'

```
In [65]: string = 'hello'
```

```
In [66]: print re.match(r'hello?', string).group()

hello
```

```
In [67]: print re.match(r'hello?', 'hell').group()

hell
```

```
In [68]: print re.match(r'hello?', 'hel')

None
```

```
In [69]: print re.match(r'hell?o?', 'hel').group()

hel
```

```
In [70]: print re.match('hell?o', 'helll')   # 'l' is repeating 3 times

None
```

```
In [71]: print re.match('hell?o', 'helllo')   # 'l' is repeating 3 times

None
```

```
In [72]: print re.match('hell?o', 'helo').group()

helo
```

```
*  - 0 or more
+  - 1 or more
?  - 0 or 1
```

Gready Search Patterns

*?, +?, ?? - GREEDY SEARCH Patterns

```
In [73]: string = '<H1>title</H1>'
```

```
In [74]: print re.match(r'<.*>', string).group()
```

```
<H1>title</H1>
```

```
In [75]: print re.match(r'<H*?>', string).group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-75-ad2e5d60ff7e> in <module>()
----> 1 print re.match(r'<H*?>', string).group()

AttributeError: 'NoneType' object has no attribute 'group'
```

```
In [76]: print re.match(r'<H1*?>', string).group()
```

```
<H1>
```

```
In [77]: print re.match(r'<.*?>', string).group()
```

```
<H1>
```

{m}

- specifies the exactly m copies of previous RE

ex: a{6} -- it maches six 'a' characters

```
In [78]: string = 'aaashique'
```

```
In [79]: re.match('a{3}shique', string).group()
```

```
Out[79]: 'aaashique'
```

```
In [80]: re.match('a{3}shique', 'aashique')
```

```
In [81]: print re.match('aa{2}shique', 'aaashique').group()
```

```
aaashique
```

```
In [82]: re.match('a{3}shique', 'aaaaaashique').group()

         ---------------------------------------------------------------------------
         AttributeError                            Traceback (most recent call last)
         <ipython-input-82-9d05772446f1> in <module>()
         ----> 1 re.match('a{3}shique', 'aaaaaashique').group()

         AttributeError: 'NoneType' object has no attribute 'group'
```

```
In [83]: re.match('aaaa{3}shique', 'aaaaaashique').group()

Out[83]: 'aaaaaashique'
```

{m,n}

- causes the resulting RE to match from m to n repitions of the preceding RE

ex: a{3,5} will match from 3 to 5 'a' characters

```
In [84]: string = 'aaashique'
```

```
In [85]: print re.match('a{2,3}shique', string).group()

         aaashique
```

```
In [86]: print re.match('a{2,3}shique',  'aashique').group()

         aashique
```

```
In [87]: print re.match('aa{2,3}shique',  'aaaashique').group()

         aaaashique
```

{m,n}? - combined regex pattern

```
In [88]: print re.match('a{1,2}?shique',  'aashique').group()

         aashique
```

```
In [89]: re.match('a{2,3}','aaaaaa').group()

Out[89]: 'aaa'
```

```
In [90]: re.search('a{2,3}',string).group()

Out[90]: 'aaa'
```

```
In [91]: re.search('a{2,3}?',string).group()   # takes 2 occurrences, due to the presen
         ce of '?'

Out[91]: 'aa'
```

\ either escapes the special characters (permittin you to match characters like '*', '?') or used to signal a special sequence

```
In [92]: string = '<H*>test<H*>'
```

```
In [93]: re.match('<H*>',string).group()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-93-b5f7ee7a2e36> in <module>()
----> 1 re.match('<H*>',string).group()

AttributeError: 'NoneType' object has no attribute 'group'
```

```
In [94]: re.match('<H\*>',string).group()
Out[94]: '<H*>'
```

```
In [95]: string = '<H?>test<H?>'
```

```
In [96]: re.match('<H\?>',string).group()
Out[96]: '<H?>'
```

[]

- used to indicate a set of characters.
- regular expression characters will lose their significance, within the [] (square) braces

ex:

```
[mnk] - will match the characters 'm', 'n' and 'k'

[a-z] - will match all characters from 'a' to 'z'

[A-Z] - will match all characters from 'A' to 'Z'

[0-9] - will match all characters from 0 to 9

[a-m] - will match all characters from 'a' to 'm'
```

```
In [97]: print re.match('h[eE]llo','hello').group()
hello
```

```
In [98]: print re.match('h[eE]llo','hEllo').group()
hEllo
```

```
In [99]:  print re.match('h[eE]llo','heEllo')

          None
```

```
In [100]: print re.match('h[eE]llo','heello')

          None
```

```
In [101]: print re.match('h[eE]*llo','heello').group()

          heello
```

```
In [102]: re.match('[a-z].*','hello').group()

Out[102]: 'hello'
```

```
In [103]: re.match('[a-z].*','hello123').group()

Out[103]: 'hello123'
```

```
In [104]: re.match('[a-z]','hello123').group()

Out[104]: 'h'
```

```
In [105]: print re.search('[a-z]$','hello123')

          None
```

```
In [106]: re.search('[0-9]$','hello123').group()

Out[106]: '3'
```

Note - special characters lose their special meaning inside sets.

To match a literal ']' inside a set, precede it with a backslash, or place it at the beginning of the set. For example, both [()[]{}] and [] () {[]} will match a parenthesis

```
In [107]: string = '<h*>test<h*>'
```

```
In [108]: re.match('<h[*]>',string).group()       # esacaping *

Out[108]: '<h*>'
```

```
In [109]: re.match('<h\*>',string).group()        # esacaping *

Out[109]: '<h*>'
```

Interview Question : Write a regular expression, to match all email IDs?

Assignment : Identify the email IDs in the resume in the following paragraph.

```
Welcome to RegExr v2.1 by gskinner.com, proudly hosted by Media Temple!

Edit the Expression & Text to see matches. Roll over matches or the expression for
 details. Undo mistakes with ctrl-z. Save Favorites & Share expressions with friend
s or the Community. Explore your results with Tools. A full Reference & Help is ava
ilable in the Library, or watch the video Tutorial.
python@programm.com
Sample text for testing:
abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 _+-.,!@#$%^&*();\/|<>"'python@gmail.com
12345 -98.7 3.141 .6180 9,000 +42
555.123.4567    +1-(800)-555-2468
foo@demo.net    bar.ba@test.co.uk
www.demo.com    http://foo.co.uk/
http://regexr.com/foo.html?q=bar
https://mediatemple.net
mediatepmple@outlook.com
mubeen.tom@hacker.com
1%453&harini_new@in.com
```

- **Popular Regular expression Generators :**
  - http://regexr.com/ (http://regexr.com/)
  - https://regex101.com/#python (https://regex101.com/#python)
  - http://www.regular-expressions.info/python.html (http://www.regular-expressions.info/python.html)
- **Popular Regular Expression Visualizers :**
  - https://regexper.com (https://regexper.com)

re.IGNORECASE

- to ignore the case (upper and lower)

In [110]:
```
regObject = re.compile('python', re.IGNORECASE)  #compiling of regex object le
ts us to reuse it

result = regObject.search('PYTHON')
print result
if result:
    print result.group()
```

```
<_sre.SRE_Match object at 0x0396DD40>
PYTHON
```

In [111]:
```
reg = re.compile('python', re.I)   # Both re.I and re.IGNORECASE work in same
 way

result = reg.search('PyThOn')
print result
if result:
    print result.group()
```

```
<_sre.SRE_Match object at 0x0396DC28>
PyThOn
```

In [112]:
```
print re.search('python', 'PyThOn', re.I).group()     # Alternative method
```

```
PyThOn
```

re.DOTALL

- special character match any character at all, includinh a newline

In [113]:
```
string = 'Today is Friday.\n Tomarrow is morning'
print string
```

```
Today is Friday.
 Tomarrow is morning
```

In [114]:
```
reg = re.compile('.*')

print reg.search(string).group()      # Observe that only first line is matche
d
```

```
Today is Friday.
```

In [115]:
```
reg = re.compile('.*', re.DOTALL)

print reg.search(string).group()      # Now, all the lines will be matched
```

```
Today is Friday.
 Tomarrow is morning
```

In [116]:
```
print re.search('.*', string, re.DOTALL).group()     # ALTERNATIVELY
```

```
Today is Friday.
 Tomarrow is morning
```

Grouping

```
\w  - presence of Alphabet
\W  - absence of Alphabet
\d  - presence of digit
\D  - absence of digit
\s  - presence of White-space
\S  - absence of white-space
```

In [117]:
```python
print re.search('\w', 'udhay prakash').group()
```
```
u
```

In [118]:
```python
print re.search('\w*', 'udhay prakash').group()
```
```
udhay
```

In [119]:
```python
print re.search('(\w)', 'udhay prakash').group()
```
```
u
```

In [120]:
```python
print re.search('(\w*)', 'udhay prakash').group()
```
```
udhay
```

In [121]:
```python
print re.search('(\w*)', 'udhay prakash').group(0)
```
```
udhay
```

In [122]:
```python
print re.search('(\w*) (\w*)', 'udhay prakash').group()
```
```
udhay prakash
```

In [123]:
```python
print re.search('(\w*) (\w*)', 'udhay prakash').group(0)  # group(0) is same a
s group()
```
```
udhay prakash
```

In [124]:
```python
print re.search('(\w*) (\w*)', 'udhay prakash').group(1)
```
```
udhay
```

In [125]:
```python
print re.search('(\w*) (\w*)', 'udhay prakash').group(2)
```
```
prakash
```

In [126]:
```python
print re.search('(\w*)(\W)(\w*)', 'udhay prakash').group()
```
```
udhay prakash
```

In [127]:
```python
print re.search('(\w*)(\W)(\w*)', 'udhay prakash').group(3)
```
```
prakash
```

```
In [128]: print re.search('(\w*)(\s)(\w*)', 'udhay prakash').group()

          udhay prakash
```

Perl based grouping pattern

`(?P<name>)`

```
In [129]: m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Barack Obama" )

In [130]: m.group()

Out[130]: 'Barack Obama'

In [131]: m.group(0)

Out[131]: 'Barack Obama'

In [132]: m.group(2)

Out[132]: 'Obama'

In [133]: m.group('first_name')

Out[133]: 'Barack'

In [134]: m.group('last_name')

Out[134]: 'Obama'

In [135]: first_name    # Observe that those identifiers can't be used outside

          ---------------------------------------------------------------------------
          NameError                                 Traceback (most recent call last)
          <ipython-input-135-c57da601b80c> in <module>()
          ----> 1 first_name

          NameError: name 'first_name' is not defined

In [136]: re.match(r'(..)+', 'alb2cs').group()

Out[136]: 'alb2cs'
```

NOTE: If a group matches multiple times, only the last match is accessible

```
In [137]: re.match(r'(..)+', 'alb2cs').group(0)

Out[137]: 'alb2cs'
```

```
In [138]: re.match(r'(..)+', 'alb2cs').group(1)

Out[138]: 'cs'
```

```
In [139]: re.match(r'(..)+', 'alb2cs').group(2)

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-139-78100e216102> in <module>()
----> 1 re.match(r'(..)+', 'alb2cs').group(2)

IndexError: no such group
```

Assignment : Try replacing match with search in the below expression, and reevaluate

```
    re.match(r'(..)+', 'alb2cs').group(1)
```

ktg cl19