Python Programming

by Narendra Allam

Chapter 5

Functions

Topics Covering

- Purpose of a function
- Defining a function
- Calling a function
- Function parameter passing
- Formal arguments
- Actual arguments
- Positional arguments
- Keyword arguments
- Variable arguments
- Variable keyword arguments
- Use-Case *args, **kwargs
- Function call stack
    - locals()
    - globals()
    - Stackframe
- Call-by-object-reference
- Shallow copy - copy.copy()
- Deep copy - copy.deepcopy()
- recursion
- Passing functions to functions
- Defining functions within functions
- Returning function from a function
- Closure and Capturing
- Decorators
- Genrators
- Closures
- Interview Questions
- Exercises

Purpose:

- Maximizing code reuse and minimizing redundancy
- Procedural decomposition which makes code maintainable

Lets start with factorial program

In[]

```
n = 5
f = 1
for x in range(1, n+1):
    f = f * x
print(f)
```

```
120
```

What if, we want to write nCr program?

```
nCr = n!/(n-r)!*r!
```

Three times we need to compute factorial, for 'n', 'n-r' and 'r'. Do we have to copy the above logic at all the three places? Let's assume we copied the logic.

In[]

```


n = 5
r = 2


nfact = 1
for x in range(1, n+1):
    nfact = nfact * x


nrfact = 1
for x in range(1, n-r+1):
    nrfact = nrfact * x


rfact = 1
for x in range(1, r+1):
    rfact = rfact * x


print 'nCr: ',  nfact/(nrfact*rfact)
```

```
nCr:  10
```

There are 3 problems here

- This increases code size, new variables are required.
- If there is any bug hidden, bug will be replicated, and we will have to fix at all the places
- If we want to enhance the logic, again we have to do it at all the places

What if, we are able to name the block of reusable code and, can execute this block, by simply calling its name whereever it is necessary?

Functions

- Function is a block of reusable code, which performs a task.
- Functions can take data and return data, but it is optional.
- Functions are first class objects in python.

Syntax:

```
def func_name(Param1, Param2, ...):
    statements
    return value(s)
```

Example:

```
def add(x, y):
    z = x + y
    return z
```

```
In[]  def add(x, y): # function definition
          z = x + y
          return z

      result = add(3, 4) # function call
      print result
```

```
7
```

```
In[] # computing nCr by reusing the factorial program
     def fact(n):
         f = 1
         for x in range(1, n+1):
             f = f * x
         return f

     n = 5
     r = 3
     res = fact(n)/(fact(n-r)*fact(r))
     print 'nCr = ', res
```

```
nCr =  10
```

Recieving parameters and returning values is just optional

```
In[] # function which doesn't take anything and doesn't return  anything
     def greet():
         print 'Hello, How are you today?'

     greet()
```

```
Hello, How are you today?
```

Note: In python it is allowed to epect a value when a function doesn't return anything. We get None.

```
In[] res = greet()
     print res
```

```
Hello, How are you today?
None
```

Function Parameter passing

```
In[] # definition of function add
     def add(x, y, z): # formal parameters
         s = x + y + z
         return s

     a = 2
     b = 4
     # function call
     final_sum = add(a, 3, b)# actual arguments
     print final_sum
```

9

In the above code

- a, 3, b are actual arguments
- x, y, z are formal parameters

```
In[]  add(4,5,6)
```

```
Output: 15
```

```
In[]  # positional arguments are mandatory arguments,
      # Arguments will be recievd by the formal arguments
      # in the same order of actual arguments.
      # we cannot skip passing them.
      def add(x, y, z):
          s = x + y + z
          return s

      print add(3, 4, 5) # positional arguments
```

```
12
```

Keyword Arguments

```
In[]  add(2, z=4, y=5) # keyword arguments
```

```
Output: 11
```

```
In[]  add(2, y=5, 10) # is not possible
```

```
  File "<ipython-input-11-aff872c2f5b3>", line 1
    add(2, y=5, 10) # is not possible
SyntaxError: non-keyword arg after keyword arg
```

Default Arguments

```
In[]  def add(x, y, z=5): # default arguments
          s = x + y + z
          return s
```

```
In[]  # z takes default value, 5
      print add(3, 4)

      # z's default value replaced by the actucal argument, 10
      print add(4, 3, 10)
```

```
12
17
```

```
In[]  def add(x, y=0, z=0): # deafult arguments
          s = x + y + z
          return s

      print add(4)
```

```
4
```

```
In[]  add(4, 5)
```

```
Output: 9
```

```
In[]  print add(4, z=5)
```

```
9
```

***** non-default arguments must not follow default arguments**

```
In[]  def add(x=0, y, z=0):
          s = x + y + z
          return s

      print add(4, 5)
```

```
  File "<ipython-input-17-991b2bfb6567>", line 1
    def add(x=0, y, z=0):
SyntaxError: non-default argument follows default argument
```

```
In[]  def add(x=0, y=0, z=0): # deafult arguments
          s = x + y + z
          return s
      print add()
```

```
0
```

```
In[]  print add(z=5, y=4)# if we want pass value to z
```

```
9
```

Variable arguments

usecase : sum(), avg()

```
In[]  def add(*args):
          print type(args)
          s = 0
          for x in args:
              s = s + x
          return s
```

Note: Varaible arguments are sent to the function as a tuple

```
In[]  add(2, 3, 4.0, 5, 6, 7.5, 8, 9)

      <type 'tuple'>

Output: 44.5
```

```
In[]  add(8, 9)

      <type 'tuple'>

Output: 17
```

```
In[]  add()

      <type 'tuple'>

Output: 0
```

Ternary Operator

```
Syntax:

result = val_1 if (condition) else val_2
```

```
In[]  y = 7
      x = 20 if y < 0 else 30
      print x

      30
```

Variable Keyword arguments

- Variable keyword arguments are sent to a function as a dict
- Easy to maintain API, we can easily incorporate new changes in the implementation of a function, without changing its signature, thus without breaking applications.

```
In[] def add(**kwargs):

         for var, val in kwargs.items():
             print var,'->', val
```

```
In[] add(a=20, b=30, c=40) # ==> add({'a':20, 'b':30, 'c':40})
```

```
a -> 20
c -> 40
b -> 30
```

```
In[] add()
```

- Alternate syntax to create a dict()

```
In[] d = dict(a=20, b=30, c=40)
     print d
```

```
{'a': 20, 'c': 40, 'b': 30}
```

```
In[] def submit_form(name, addr, mobile, email):
         print 'Storing data in the database: {}, {}'.format(name, addr)
         print 'SMS: {} your application has been  submited'.format(mobil
     e)
         print 'EMAIL: {} your application has been  submited'.format(ema
     il)
```

```
In[] submit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.com')
```

```
Storing data in the database: Ashok, Hyd
SMS: 0 your application has been  submited
EMAIL: ashok@hotmail.com your application has been submited
```

```
In[]  def submit_form(name, addr, citizen='IND', *args, **kwargs):
          """
          syntax : submit_form(name, addr, nationality='Indian')
          param name(mandatory): str name of the candidate
          param addr(optional): str address
          param nationality(deafult='IND'): str nationality
          param mobile(optional): str valid phone
          param email(optional): str valid email
          """
          print kwargs
          print 'Storing data in the database: {}, {}, {}'.format(name, a
      ddr, citizen)

          if 'mobile' in kwargs:
              print 'SMS: {} your application has been submited'.format(k
      wargs['mobile'])
          if 'email' in kwargs:
              print 'EMAIL: {} your application has been submited'.format
      (kwargs['email'])
```

```
In[]  submit_form('Ashok', 'hyderabad', mobile='9875976554')
```

```
{'mobile': '9875976554'}
Storing data in the database: Ashok, hyderabad, IND
SMS: 9875976554 your application has been  submited
```

```
In[]  submit_form('jhon', 'Delhi', email='jhon@gmail.com')
```

```
{'email': 'jhon@gmail.com'}
Storing data in the database: jhon, Delhi, IND
EMAIL: jhon@gmail.com your application has been submited
```

```
In[]  submit_form('Karthik',
                         'Chennai',
                         mobile='9875976554',
                         email='karthik@gmail.com')
```

```
{'mobile': '9875976554', 'email': 'karthik@gmail.com'}
Storing data in the database: Karthik, Chennai, IND
SMS: 9875976554 your application has been  submited
EMAIL: karthik@gmail.com your application has been submited
```

```
In[]  submit_form('Naren', 'Hyd')
```

```
{}
Storing data in the database: Naren, Hyd, IND
```

In[]
```
def simple_interest(principle, **kwargs):

    duration = kwargs.get('duration', 12)
    rate = kwargs.get('rate', 0.12)

    return principle*duration*rate
```

In[]
```
simple_interest(2100000, duration=24)
```

Output:
```
6048000.0
```

Note: Varaible keyword arguments are sent to the function as a dict.

In[]
```
add(x=2, y=3, z=40)
```

```
y -> 3
x -> 2
z -> 40
```

In[]
```
d = {'x': 20, 'y':30, 'z':40}
print d
```

```
{'y': 30, 'x': 20, 'z': 40}
```

In[]
```
d = dict(x=20, y=30, z=40)
print d
```

```
{'y': 30, 'x': 20, 'z': 40}
```

```
In[] def fun(a, b, c=10, d=20, *args, **kwargs):
         s = a + b + c + d
         print '--------------------'
         print 'Positional arguments'
         print '--------------------'
         print 'a = ', a, ' b = ', b

         print '--------------------'
         print 'Default arguments'
         print '--------------------'
         print 'c = ', c, ' d = ', d

         print '--------------------'
         print 'Variable arguments'
         print '--------------------'
         print args

         print '--------------------'
         print 'Variable Keyword arguments'
         print '--------------------'
         print kwargs

         return s
```

- **a, b - positional arguments**
- **c, d - default arguments**
- **args - variable arguments**
- **kwargs - variable keyword arguments**

```
In[] res = fun(2, 3, 4, 5, 6, 7, 8, p=10, q=20)
```

```
--------------------
Positional arguments
--------------------
a =  2  b =  3
--------------------
Default arguments
--------------------
c =  4  d =  5
--------------------
Variable arguments
--------------------
(6, 7, 8)
--------------------
Variable Keyword arguments
--------------------
{'q': 20, 'p': 10}
```

In[]
```
res = fun(2, 3, 5, 6)
```

```
--------------------
Positional arguments
--------------------
a =  2  b =  3
--------------------
Default arguments
--------------------
c =  5  d =  6
--------------------
Variable arguments
--------------------
()
--------------------
Variable Keyword arguments
--------------------
{}
```

In[]
```
res = fun(2,3, k=10)
```

```
--------------------
Positional arguments
--------------------
a =  2  b =  3
--------------------
Default arguments
--------------------
c =  10  d =  20
--------------------
Variable arguments
--------------------
()
--------------------
Variable Keyword arguments
--------------------
{'k': 10}
```

```
In[] res = fun(2,3, d=10)
```

```
--------------------
Positional arguments
--------------------
a =  2  b =  3
--------------------
Default arguments
--------------------
c =  10  d =  10
--------------------
Variable arguments
--------------------
()
--------------------
Variable Keyword arguments
--------------------
{}
```

```
In[] res = fun(4,5)
```

```
--------------------
Positional arguments
--------------------
a =  4  b =  5
--------------------
Default arguments
--------------------
c =  10  d =  20
--------------------
Variable arguments
--------------------
()
--------------------
Variable Keyword arguments
--------------------
{}
```

```
In[] res = fun()
```

```
-----------------------------------------------------------------
----------
TypeError                                 Traceback (most recent c
all last)
<ipython-input-47-0db02b049119> in <module>()
----> 1 res = fun()

TypeError: fun() takes at least 2 arguments (0 given)
```

Scope

In the below program n and x in main() function are completely different from x and n are in fun(). x and n in main() function are only accessible for main function. When we pass n to fun(), the value of n will have a new name in new locality which. This called locality of reference. x, n inside fun() are brand new x and n. As long as execution control is in fun() it has its own set of local names.

Globals and Locals

In[]

```
g = 9.8

def fun(n):
    x = 50
    y = 90
    n = n*10
    print '---- in side  fun()-------'
    print '----- locals() ----------'
    print locals()
    print '----- globals() ----------'
    print globals()

def main():
    x = 20
    n = 30
    fun(n)
    print '---- in side  main()-------'
    print '----- locals() ----------'
    print locals()
    print '----- globals()  ----------'
    print globals()


main()
```

```
---- in side  fun()-------
----- locals() ----------
{'y': 90, 'x': 50, 'n': 300}
----- globals() ----------
{'_': 0, '_builtin__': <module '__builtin__' (built-in)>, '_i2':
u"# NCR Program\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range
(1, n+1):\n    nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in r
ange(1, n-r+1):\n    nrfact = nrfact * x\n    \n# r!    \nrfact =
1\nfor x in range(1, r+1):\n    rfact = rfact * x\n    \n    \npri
nt 'nCr: ', nfact/(nrfact*rfact)", 'fact': <function fact at 0x103
5a0398>, 'quit': <IPython.core.autocall.ZMQExitAutocall object at
0x1034bfa90>, 'add': <function add at 0x1035a0578>, 'In': ['',  u'n
= 5\nf = 1\nfor x in  range(1, n+1):\n    f = f * x\nprint(f)', u"#
NCR Program\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range(1,
n+1):\n    nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in range
```

```
(1, n-r+1):\n     nrfact = nrfact * x\n     \n# r!     \nrfact = 1\nf
or x in range(1, r+1):\n     rfact = rfact * x\n     \n     \nprint '
```

```
nCr: ', nfact/(nrfact*rfact)", u'def add(x, y): # function definit
ion\n    z = x + y\n    return z\n\nresult = add(3, 4) # function
call\nprint result', u"# computing nCr by reusing the factorial pr
ogram\ndef fact(n): \n    f = 1 \n    for x in range(1, n+1): \n
f = f * x\n return f\n\nn = 5\nr = 3\nres = fact(n)/(fact(n-r)*
fact(r))\nprint 'nCr = ', res", u"# function which doesn't take an
ything and doesn't return anything\ndef greet():\n    print 'Hello
, How are you today?'\n\ngreet()", u'res = greet()\nprint res', u'
# definition of function add\ndef add(x, y, z): # formal parameter
s\n s = x + y + z\n return s\n\na = 2\nb = 4\n# function cal
l\nfinal_sum = add(a, 3, b)# actual arguments\nprint final_sum', u
'add(4,5,6)', u'# positional arguments are mandatory arguments, \n
# Arguments will be recievd by the formal arguments \n# in the sam
e order of actual arguments.\n# we cannot skip passing them.\ndef
add(x, y, z): \n    s = x + y + z\n    return s\n\nprint add(3, 4,
5) # positional arguments', u'add(2, z=4, y=5) # keyword arguments
', u'add(2, y=5, 10) # is not possible', u'def add(x, y, z=5): # d
efault arguments\n    s = x + y + z\n    return s', u"# z takes de
fault value, 5\nprint add(3, 4)\n\n# z's default value replaced by
the actucal argument, 10\nprint add(4, 3, 10)", u'def add(x, y=0,
z=0): # deaful arguments\n    s = x + y + z\n    return s\n\nprin
t add(4)', u'add(4, 5)', u'print add(4, z=5)', u'def add(x=0, y,  z
=0): \n    s = x + y + z\n    return s\n\nprint add(4, 5)', u'def
add(x=0, y=0, z=0): # deaful arguments\n    s = x + y + z\n    re
turn s\nprint add()', u'print add(z=5, y=4)# if we want pass value
to z', u'def add(*args):\n    print type(args)\n    s = 0\n    for
x in args:\n        s = s + x\n    return s', u'add(2, 3, 4.0, 5,
6, 7.5, 8, 9)', u'add(8, 9)', u'add()', u'y = 7\nx = 20 if y < 0 e
lse 30\nprint x', u"def add(**kwargs):\n\n    for var, val in kwar
gs.items():\n        print var,'->', val", u"add(a=20, b=30, c=40)
# ==> add({'a':20, 'b':30, 'c':40})", u'add()', u'd = dict(a=20, b
=30, c=40)\nprint d', u"def submit_form(name, addr, mobile, email)
:\n print 'Storing data in the database: {}, {}'.format(name, a
ddr)\n print 'SMS: {} your application has been submited'.forma
t(mobile)\n print 'EMAIL: {} your application has been submited
'.format(email)", u"submit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.
com')", u'def submit_form(name, addr, citizen=\'IND\', *args, **kw
args):\n    """\n    syntax : submit_form(name, addr, nationality=
\'Indian\')\n    param name(mandatory): str name of the candidate\
n    param addr(optional): str address\n    param nationality(deaf
ult=\'IND\'): str nationality\n    param mobile(optional): str val
id phone \n    param email(optional): str valid email\n    """\n
print kwargs\n    print \'Storing data in the database: {}, {}, {}
\'.format(name, addr, citizen)\n    \n    if \'mobile\' in kwargs:
\n print \'SMS: {} your application has been submited\'.for
mat(kwargs[\'mobile\'])\n    if \'email\' in kwargs:\n        prin
t \'EMAIL: {} your application has been submited\'.format(kwargs[\
'email\'])\n   ',  u"submit_form('Ashok',  'hyderabad',  mobile=
'9875976554')", u"submit_form('jhon', 'Delhi', email='jhon@gmail.c
om')", u"submit_form('Karthik', \n                'Chennai', \n
mobile='9875976554', \n                email='karthik@gmail.com
')", u"submit_form('Naren', 'Hyd')", u"def simple_interest(princip
le, **kwargs):\n\n    duration = kwargs.get('duration', 12)\n    r
ate = kwargs.get('rate', 0.12)\n\n     return principle*duration*ra
```

```
te ", u'simple_interest(2100000, duration=24)', u'add(x=2, y=3, z=
40)', u"d = {'x': 20, 'y':30, 'z':40}\nprint d", u'd = dict(x=20,
y=30, z=40)\nprint d', u"def fun(a, b, c=10, d=20, *args,  **kwargs
):\n    s = a + b + c + d\n    print '--------------------'\n    pr
int 'Positional arguments'\n    print '--------------------'\n    p
rint 'a = ', a, ' b = ', b\n    \n    print '--------------------'\
n    print 'Default arguments'\n    print '--------------------'\n
print 'c = ', c, ' d = ', d\n    \n    print '--------------------'
\n    print 'Variable arguments'\n    print '--------------------'\
n    print args\n\n    print '--------------------'\n    print 'Var
iable Keyword arguments'\n    print '--------------------' \n    pr
int kwargs\n    \n    return s", u'res = fun(2, 3, 4, 5, 6, 7, 8,
p=10, q=20)', u'res = fun(2, 3, 5, 6)', u'res = fun(2,3, k=10)',  u
'res = fun(2,3, d=10)', u'res = fun(4,5)', u'res = fun()', u"g =  9
.8\n\ndef fun(n):\n    x = 50\n    y = 90\n    n = n*10\n    print
'---- in side fun()-------'\n    print '----- locals()  ----------'
\n    print locals()\n    print '----- globals() ----------'\n
print globals()\n\ndef main():\n    x = 20\n    n = 30\n    fun(n)
\n    print '---- in side main()-------'\n    print '----- locals(
) ----------'\n    print locals()\n    print '----- globals()  ----
------'\n    print globals()\n\n    \nmain()"], '_i': u'res = fun(
)', 'main': <function main at 0x103d335f0>, '_22': 17, '_doc_':
'Automatically created module for IPython interactive  environment'
, '_21': 44.5, 'final_sum': 9, 'submit_form': <function submit_for
m at 0x1035a0668>, 'nrfact': 6, 'nfact': 120, '_sh': <module 'IPyt
hon.core.shadowns'  from  '/anaconda/lib/python2.7/site-packages/IPy
thon/core/shadowns.pyc'>, 'b': 4, 'simple_interest': <function sim
ple_interest at 0x1035a0488>, 'd': {'y': 30, 'x': 20, 'z': 40}, 'f
': 120, '_8': 15, '_23': 0, '_i13': u"# z takes default value, 5\n
print add(3, 4)\n\n# z's default value replaced by the actucal arg
ument, 10\nprint add(4, 3, 10)", '_i12': u'def add(x, y, z=5): # d
efault arguments\n    s = x + y + z\n    return s', '_15': 9, '_i1
0': u'add(2, z=4, y=5) # keyword arguments', '_i17': u'def  add(x=0
, y, z=0): \n    s = x + y + z\n    return s\n\nprint add(4, 5)',
'_i16': u'print add(4, z=5)', '_i15': u'add(4, 5)', '_i14': u'def
add(x, y=0, z=0): # deaful arguments\n    s = x + y + z\n    retu
rn s\n\nprint add(4)', 'fun': <function fun at 0x1035c9cf8>, 'x':
30, '_i19': u'print add(z=5, y=4)# if we want pass value to z', '_
i18': u'def add(x=0, y=0, z=0): #  deaful arguments\n    s = x + y
+ z\n    return s\nprint add()', '_oh': {37: 6048000.0, 8: 15,  10:
11, 15: 9, 21: 44.5, 22: 17, 23: 0}, 'Out': {37: 6048000.0, 8:  15,
10: 11, 15: 9, 21: 44.5, 22: 17, 23: 0}, '_dh': [u'/Users/munna/Go
ogle Drive/Latest Notebooks'], 'result': 7, '_iii': u'res = fun(2,
3, d=10)', 'n': 5, 'rfact': 2, '_i9': u'# positional arguments are
mandatory arguments, \n# Arguments will be recievd by the formal a
rguments \n# in the same order of actual arguments.\n# we cannot s
kip passing them.\ndef add(x, y, z): \n    s = x + y + z\n    retu
rn s\n\nprint add(3, 4, 5) # positional arguments', '_i8':  u'add(4
,5,6)', '_i7': u'# definition of function add\ndef add(x, y, z): #
formal parameters\n    s = x + y + z\n    return s\n\na = 2\nb =  4
\n# function call\nfinal_sum = add(a, 3, b)# actual arguments\npri
```

```
nt final_sum', 'res': 39, '_i5': u"# function which doesn't take a
nything and doesn't return anything\ndef greet():\n    print 'Hell
o, How are you today?'\n\ngreet()", '_i4': u"# computing nCr by  re
```

```
using the factorial program\ndef fact(n): \nf = 1 \n        for x
in range(1, n+1): \n        f = f * x\n    return f\n\nn = 5\nr =
3\nres = fact(n)/(fact(n-r)*fact(r))\nprint 'nCr = ', res", '_i3':
u'def add(x, y): # function definition\n    z = x + y\n    return
z\n\nresult = add(3, 4) # function call\nprint result', '_i11': u'
add(2, y=5, 10) # is not possible', '_i1': u'n = 5\nf = 1\nfor x i
n range(1, n+1):\n    f = f * x\nprint(f)', '_i44': u'res = fun(2,
3, k=10)', 'r': 3, 'exit': <IPython.core.autocall.ZMQExitAutocall
object at 0x1034bfa90>, 'get_ipython': <bound method ZMQInteractiv
eShell.get_ipython of <ipykernel.zmqshell.ZMQInteractiveShell obje
ct at 0x103478350>>, '_i28': u'd = dict(a=20, b=30, c=40)\nprint d
', '_i29': u"def submit_form(name, addr, mobile, email):\n    prin
t 'Storing data in the database: {}, {}'.format(name, addr)\n    p
rint 'SMS: {} your application has been submited'.format(mobile)\n
print 'EMAIL: {} your application has been submited'.format(email)
", '_i26': u"add(a=20, b=30, c=40) # ==> add({'a':20, 'b':30, 'c':
40})", '_i27': u'add()', '_i24': u'y = 7\nx = 20 if y < 0 else 30\
nprint x', '_i25': u"def add(**kwargs):\n\n    for var, val in kwa
rgs.items():\n        print var,'->', val", '_i22': u'add(8, 9)',
'_i23': u'add()', '_i20': u'def add(*args):\n    print type(args)\
n    s = 0\n    for x in args:\n        s = s + x\n    return s',
'_i21': u'add(2, 3, 4.0, 5, 6, 7.5, 8, 9)', '_i47': u'res = fun()'
, '_i48': u"g = 9.8\n\ndef fun(n):\n    x = 50\n    y = 90\n    n
= n*10\n    print '---- in side fun()-------'\n    print '----- lo
cals() ----------'\n    print locals()\n    print '----- globals()
----------'\n    print globals()\n\ndef main():\n    x = 20\n    n
= 30\n    fun(n)\n    print '---- in side main()-------'\n    prin
t '----- locals() ----------'\n    print locals()\n    print '----
- globals() ----------'\n    print globals()\n\n    \nmain()", '
builtins ': <module ' builtin ' (built-in)>, '_i45': u'res = fu
n(2,3, d=10)', '_i46': u'res = fun(4,5)', '_ih': ['', u'n = 5\nf =
1\nfor x in range(1, n+1):\n    f = f * x\nprint(f)', u"# NCR Prog
ram\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range(1, n+1):\n
nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in range(1, n-r+1):
\n  nrfact = nrfact * x\n    \n# r!    \nrfact = 1\nfor x in ran
ge(1, r+1):\n rfact = rfact * x\n \n \nprint 'nCr: ', nfa
ct/(nrfact*rfact)", u'def add(x, y): # function definition\n z
= x + y\n return z\n\nresult = add(3, 4) # function call\nprint
result', u"# computing nCr by reusing the factorial program\ndef f
act(n): \n f = 1 \n for x in range(1, n+1): \n f =  f
* x\n    return f\n\nn = 5\nr = 3\nres = fact(n)/(fact(n-r)*fact(r
))\nprint 'nCr = ', res", u"# function which doesn't take anything
and doesn't return anything\ndef greet():\n    print 'Hello, How a
re you today?'\n\ngreet()", u'res = greet()\nprint res', u'# defin
ition of function add\ndef add(x, y, z): # formal parameters\n
s = x + y + z\n return s\n\na = 2\nb = 4\n# function call\nfina
l_sum = add(a, 3, b)# actual arguments\nprint final_sum', u'add(4,
5,6)', u'# positional arguments are mandatory arguments, \n# Argum
ents will be recievd by the formal arguments \n# in the same order
of actual arguments.\n# we cannot skip passing them.\ndef add(x, y
, z): \n    s = x + y + z\n    return s\n\nprint add(3, 4, 5) # po
```

```
sitional arguments', u'add(2, z=4, y=5) # keyword arguments', u'ad
d(2, y=5, 10) # is not possible', u'def add(x, y, z=5): # default
arguments\n    s = x + y + z\n    return s', u"# z takes default v
```

```
alue, 5\nprint add(3, 4)\n\n# z's default value replaced by the ac
tucal argument, 10\nprint add(4, 3, 10)", u'def add(x, y=0,  z=0):
# deafult arguments\n   s = x + y + z\n    return s\n\nprint add(
4)', u'add(4, 5)', u'print add(4, z=5)', u'def add(x=0, y, z=0): \
n   s = x + y + z\n    return s\n\nprint add(4, 5)', u'def add(x=
0, y=0, z=0): # deafult arguments\n s = x + y + z\n return  s
\nprint add()', u'print add(z=5, y=4)# if we want pass value to  z'
, u'def add(*args):\n    print type(args)\n    s = 0\n    for x in
args:\n        s = s + x\n    return s', u'add(2, 3, 4.0, 5, 6, 7.
5, 8, 9)', u'add(8, 9)', u'add()', u'y = 7\nx = 20 if y < 0 else 3
0\nprint x', u"def add(**kwargs):\n\n for var, val in kwargs.it
ems():\n print var,'->', val", u"add(a=20, b=30, c=40) # ==
> add({'a':20, 'b':30, 'c':40})", u'add()', u'd = dict(a=20, b=30,
c=40)\nprint d', u"def submit_form(name, addr, mobile, email):\n
print 'Storing data in the database: {}, {}'.format(name, addr)\n
print 'SMS: {} your application has been submited'.format(mobile)\
n    print 'EMAIL: {} your application has been submited'.format(e
mail)", u"submit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.com')", u'
def submit_form(name, addr, citizen=\'IND\', *args, **kwargs):\n
"""\nsyntax  : submit_form(name, addr, nationality=\'Indian\')\
n    param name(mandatory): str name of the candidate\n    param a
ddr(optional): str address\n    param nationality(deafult=\'IND\')
: str nationality\n    param mobile(optional): str valid phone \n
param email(optional): str valid email\n    """\nprint   kwargs\
n    print \'Storing data in the database: {}, {}, {}\'.format(nam
e, addr, citizen)\n    \n    if \'mobile\' in kwargs:\n        pri
nt \'SMS: {} your application has been submited\'.format(kwargs[\'
mobile\'])\n    if \'email\' in kwargs:\n        print \'EMAIL: {}
your application has been submited\'.format(kwargs[\'email\'])\n
', u"submit_form('Ashok', 'hyderabad', mobile='9875976554')", u"su
bmit_form('jhon', 'Delhi', email='jhon@gmail.com')", u"submit_form
('Karthik', \n                  'Chennai', \n                     m
obile='9875976554', \n                  email='karthik@gmail.com'
)", u"submit_form('Naren', 'Hyd')", u"def simple_interest(principl
e, **kwargs):\n\n    duration = kwargs.get('duration', 12)\n    ra
te = kwargs.get('rate', 0.12)\n\n    return principle*duration*rat
e ", u'simple_interest(2100000, duration=24)', u'add(x=2, y=3, z=4
0)', u"d = {'x': 20, 'y':30, 'z':40}\nprint d", u'd = dict(x=20,  y
=30, z=40)\nprint d', u"def fun(a, b, c=10, d=20, *args,  **kwargs)
:\n    s = a + b + c + d\n    print '--------------------'\n    pri
nt 'Positional arguments'\n    print '--------------------'\n    pr
int 'a = ', a, ' b = ', b\n    \n    print '--------------------'\n
print 'Default arguments'\n    print '--------------------'\n    pr
int 'c = ', c, ' d = ', d\n    \n    print '--------------------'\n
print 'Variable arguments'\n    print '--------------------'\n    p
rint args\n\n    print '--------------------'\n    print 'Variable
Keyword arguments'\n    print '--------------------' \n    print kw
args\n    \n    return s", u'res = fun(2, 3, 4, 5, 6, 7, 8, p=10,
q=20)', u'res = fun(2, 3, 5, 6)', u'res = fun(2,3, k=10)', u'res  =
fun(2,3, d=10)', u'res = fun(4,5)', u'res = fun()', u"g = 9.8\n\nd
ef fun(n):\n    x = 50\n    y = 90\n    n = n*10\n    print '----
```

```
in side fun()-------'\n    print '----- locals() ----------'\n
print locals()\n    print '----- globals() ----------'\n    print
globals()\n\ndef main():\n    x = 20\n    n = 30\n    fun(n)\n
```

```
print '---- in side main()-------'\n    print '----- locals() ----
------'\n    print locals()\n    print '----- globals() ----------
'\n    print globals()\n\n    \nmain()"], '_i40': u'd = dict(x=20,
y=30, z=40)\nprint d', '_i41': u"def fun(a, b, c=10, d=20,  *args,
**kwargs):\n    s = a + b + c + d\n    print '--------------------'
\n    print 'Positional arguments'\n    print '--------------------
'\n    print 'a = ', a, ' b = ', b\n    \n    print '-------------
------'\n    print 'Default arguments'\n    print '----------------
----'\n    print 'c = ', c, ' d = ', d\n    \n    print '---------
----------'\n    print 'Variable arguments'\n    print '----------
---------'\n    print args\n\n    print '--------------------'\n
print 'Variable Keyword arguments'\n    print '-------------------
' \n    print kwargs\n    \n    return s", '_i42': u'res = fun(2,
3, 4, 5, 6, 7, 8, p=10, q=20)', '_i43': u'res = fun(2, 3, 5,  6)',
'y': 7, '_10': 11, '__name__': '__main__', '___': 17, '_': 6048000
.0, 'a': 2, 'g': 9.8, 'greet': <function greet at 0x1035a00c8>, '_
i39': u"d = {'x': 20, 'y':30, 'z':40}\nprint d", '_i38':  u'add(x=2
, y=3, z=40)', '_37': 6048000.0, '_ii': u'res = fun(4,5)', '_i6':
u'res = greet()\nprint res', '_i31': u'def submit_form(name, addr,
citizen=\'IND\',  *args,  **kwargs):\n  """\n  syntax  :  submit_f
orm(name, addr, nationality=\'Indian\')\n     param name(mandatory)
: str name of the candidate\n     param addr(optional): str address
\n    param nationality(deafult=\'IND\'): str nationality\n    par
am mobile(optional): str valid phone \n    param email(optional):
str valid email\n    """\n    print kwargs\n    print \'Storing da
ta in the database: {}, {}, {}\'.format(name, addr, citizen)\n
\n    if \'mobile\' in kwargs:\n      print \'SMS: {} your appli
cation has been submited\'.format(kwargs[\'mobile\'])\n  if  \'em
ail\' in kwargs:\n        print \'EMAIL: {} your application has b
een   submited\'.format(kwargs[\'email\'])\n   ',   '_i30':   u"sub
mit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.com')", '_i33': u"submi
t_form('jhon', 'Delhi', email='jhon@gmail.com')", '_i32':  u"submit
_form('Ashok', 'hyderabad', mobile='9875976554')", '_i35': u"submi
t_form('Naren', 'Hyd')", '_i34': u"submit_form('Karthik', \n
'Chennai', \n                    mobile='9875976554', \n
email='karthik@gmail.com')", '_i37': u'simple_interest(2100000, du
ration=24)', '_i36': u"def simple_interest(principle,  **kwargs):\n
\n    duration = kwargs.get('duration', 12)\n    rate = kwargs.get
('rate', 0.12)\n\n    return principle*duration*rate "}
---- in side main()-------
----- locals() ----------
{'x': 20, 'n': 30}
----- globals() ----------
{'_': 0, '_builtin__': <module '__builtin__' (built-in)>, '_i2':
u"# NCR Program\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range
(1, n+1):\n    nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in r
ange(1, n-r+1):\n    nrfact = nrfact * x\n    \n# r!    \nrfact =
1\nfor x in range(1, r+1):\n    rfact = rfact * x\n    \n    \npri
nt 'nCr: ', nfact/(nrfact*rfact)", 'fact': <function fact at 0x103
5a0398>, 'quit': <IPython.core.autocall.ZMQExitAutocall object at
0x1034bfa90>, 'add': <function add at 0x1035a0578>, 'In': ['',  u'n
```

```
= 5\nf = 1\nfor x in  range(1, n+1):\n     f = f * x\nprint(f)', u"#
NCR Program\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range(1,
n+1):\n     nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in  range
```

```
(1, n-r+1):\n    nrfact = nrfact * x\n\n# r!        \nrfact = 1\nf
or x in range(1, r+1):\n  rfact = rfact * x\n      \n    \nprint '
nCr: ', nfact/(nrfact*rfact)", u'def add(x, y): # function definit
ion\n    z = x + y\n    return z\n\nresult = add(3, 4) # function
call\nprint result', u"# computing nCr by reusing the factorial pr
ogram\ndef fact(n): \n    f = 1 \n    for x in range(1, n+1): \n
f = f * x\n return f\n\nn = 5\nr = 3\nres = fact(n)/(fact(n-r)*
fact(r))\nprint 'nCr = ', res", u"# function which doesn't take an
ything and doesn't return anything\ndef greet():\n    print 'Hello
, How are you today?'\n\ngreet()", u'res = greet()\nprint res', u'
# definition of function add\ndef add(x, y, z): # formal parameter
s\n s = x + y + z\n return s\n\na = 2\nb = 4\n# function cal
l\nfinal_sum = add(a, 3, b)# actual arguments\nprint final_sum', u
'add(4,5,6)', u'# positional arguments are mandatory arguments, \n
# Arguments will be recievd by the formal arguments \n# in the sam
e order of actual arguments.\n# we cannot skip passing them.\ndef
add(x, y, z): \n    s = x + y + z\n    return s\n\nprint add(3, 4,
5) # positional arguments', u'add(2, z=4, y=5) # keyword arguments
', u'add(2, y=5, 10) # is not possible', u'def add(x, y, z=5): # d
efault arguments\n    s = x + y + z\n    return s', u"# z takes de
fault value, 5\nprint add(3, 4)\n\n# z's default value replaced by
the actucal argument, 10\nprint add(4, 3, 10)", u'def add(x, y=0,
z=0): # deafult arguments\n    s = x + y + z\n    return s\n\nprin
t add(4)', u'add(4, 5)', u'print add(4, z=5)', u'def add(x=0, y,  z
=0): \n    s = x + y + z\n    return s\n\nprint add(4, 5)', u'def
add(x=0, y=0, z=0): # deafult arguments\n    s = x + y + z\n    re
turn s\nprint add()', u'print add(z=5, y=4)# if we want pass value
to z', u'def add(*args):\n    print type(args)\n    s = 0\n    for
x in args:\n        s = s + x\n    return s', u'add(2, 3, 4.0, 5,
6, 7.5, 8, 9)', u'add(8, 9)', u'add()', u'y = 7\nx = 20 if y < 0 e
lse 30\nprint x', u"def add(**kwargs):\n\n    for var, val in kwar
gs.items():\n        print var,'->', val", u"add(a=20, b=30, c=40)
# ==> add({'a':20, 'b':30, 'c':40})", u'add()', u'd = dict(a=20, b
=30, c=40)\nprint d', u"def submit_form(name, addr, mobile, email)
:\n print 'Storing data in the database: {}, {}'.format(name, a
ddr)\n print 'SMS: {} your application has been submited'.forma
t(mobile)\n print 'EMAIL: {} your application has been submited
'.format(email)", u"submit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.
com')", u'def submit_form(name, addr, citizen=\'IND\', *args, **kw
args):\n    """\n    syntax : submit_form(name, addr, nationality=
\'Indian\')\n    param name(mandatory): str name of the candidate\
n    param addr(optional): str address\n    param nationality(deaf
ult=\'IND\'): str nationality\n    param mobile(optional): str val
id phone \n    param email(optional): str valid email\n    """\n
print kwargs\n    print \'Storing data in the database: {}, {}, {}
\'.format(name, addr, citizen)\n    \n    if \'mobile\' in kwargs:
\n print \'SMS: {} your application has been submited\'.for
mat(kwargs[\'mobile\'])\n    if \'email\' in kwargs:\n        prin
t \'EMAIL: {} your application has been submited\'.format(kwargs[\
'email\'])\n   ', u"submit_form('Ashok', 'hyderabad', mobile=
'9875976554')", u"submit_form('jhon', 'Delhi', email='jhon@gmail.c
om')", u"submit_form('Karthik', \n                   'Chennai', \n
mobile='9875976554', \n                   email='karthik@gmail.com
')", u"submit_form('Naren', 'Hyd')", u"def  simple_interest(princip
```

```
le, **kwargs):\n\n    duration = kwargs.get('duration', 12)\n    r
ate = kwargs.get('rate', 0.12)\n\n    return principle*duration*ra
te ", u'simple_interest(2100000, duration=24)', u'add(x=2, y=3, z=
40)', u"d = {'x': 20, 'y':30, 'z':40}\nprint d", u'd = dict(x=20,
y=30, z=40)\nprint d', u"def fun(a, b, c=10, d=20, *args,  **kwargs
):\n    s = a + b + c + d\n    print '-------------------'\n    pr
int 'Positional arguments'\n    print '-------------------'\n    p
rint 'a = ', a, ' b = ', b\n    \n    print '-------------------'\
n    print 'Default arguments'\n    print '-------------------'\n
print 'c = ', c, ' d = ', d\n    \n    print '-------------------'
\n    print 'Variable arguments'\n    print '-------------------'\
n    print args\n\n    print '-------------------'\n    print 'Var
iable Keyword arguments'\n    print '-------------------' \n    pr
int kwargs\n    \n    return s", u'res = fun(2, 3, 4, 5, 6, 7, 8,
p=10, q=20)', u'res = fun(2, 3, 5, 6)', u'res = fun(2,3, k=10)',  u
'res = fun(2,3, d=10)', u'res = fun(4,5)', u'res = fun()', u"g =  9
.8\n\ndef fun(n):\n    x = 50\n    y = 90\n    n = n*10\n    print
'---- in side fun()-------'\n    print '----- locals() ----------'
\n    print locals()\n    print '----- globals() ----------'\n
print globals()\n\ndef main():\n    x = 20\n    n = 30\n    fun(n)
\n    print '---- in side main()-------'\n    print '----- locals(
) ----------'\n    print locals()\n    print '----- globals() ----
------'\n    print globals()\n\n    \nmain()"], '_i': u'res = fun(
)', 'main': <function main at 0x103d335f0>, '_22': 17, '_doc_':
'Automatically created module for IPython interactive  environment'
, '_21': 44.5, 'final_sum': 9, 'submit_form': <function submit_for
m at 0x1035a0668>, 'nrfact': 6, 'nfact': 120, '_sh': <module 'IPyt
hon.core.shadowns'  from  '/anaconda/lib/python2.7/site-packages/IPy
thon/core/shadowns.pyc'>, 'b': 4, 'simple_interest': <function sim
ple_interest at 0x1035a0488>, 'd': {'y': 30, 'x': 20, 'z': 40}, 'f
': 120, '_8': 15, '_23': 0, '_i13': u"# z takes default value, 5\n
print add(3, 4)\n\n# z's default value replaced by the actucal arg
ument, 10\nprint add(4, 3, 10)", '_i12': u'def add(x, y, z=5): # d
efault arguments\n    s = x + y + z\n    return s', '_15': 9, '_i1
0': u'add(2, z=4, y=5) # keyword arguments', '_i17': u'def  add(x=0
, y, z=0): \n    s = x + y + z\n    return s\n\nprint add(4, 5)',
'_i16': u'print add(4, z=5)', '_i15': u'add(4, 5)', '_i14': u'def
add(x, y=0, z=0): # deafult arguments\n    s = x + y + z\n    retu
rn s\n\nprint add(4)', 'fun': <function fun at 0x1035c9cf8>, 'x':
30, '_i19': u'print add(z=5, y=4)# if we want pass value to z', '_
i18': u'def add(x=0, y=0, z=0): # deafult arguments\n    s = x + y
+ z\n    return s\nprint add()', '_oh': {37: 6048000.0, 8: 15,  10:
11, 15: 9, 21: 44.5, 22: 17, 23: 0}, 'Out': {37: 6048000.0, 8:  15,
10: 11, 15: 9, 21: 44.5, 22: 17, 23: 0}, '_dh': [u'/Users/munna/Go
ogle Drive/Latest Notebooks'], 'result': 7, '_iii': u'res = fun(2,
3, d=10)', 'n': 5, 'rfact': 2, '_i9': u'# positional arguments are
mandatory arguments, \n# Arguments will be recievd by the formal a
rguments \n# in the same order of actual arguments.\n# we cannot s
kip passing them.\ndef add(x, y, z): \n    s = x + y + z\n    retu
rn s\n\nprint add(3, 4, 5) # positional arguments', '_i8':  u'add(4
,5,6)', '_i7': u'# definition of function add\ndef add(x, y, z): #
formal parameters\n    s = x + y + z\n    return s\n\na = 2\nb = 4
\n# function call\nfinal_sum = add(a, 3, b)# actual arguments\npri
nt final_sum', 'res': 39, '_i5': u"# function which doesn't take  a
```

```
nything and doesn't return anything\ndef greet():\n    print 'Hell
o, How are you today?'\n\ngreet()", '_i4': u"# computing nCr by re
using the factorial program\ndef fact(n): \nf = 1 \n       for x
in range(1, n+1): \n          f = f * x\n     return f\n\nn = 5\nr =
3\nres = fact(n)/(fact(n-r)*fact(r))\nprint 'nCr = ', res", '_i3':
u'def add(x, y): # function definition\n     z = x + y\nreturn
z\n\nresult = add(3, 4) # function call\nprint result', '_i11': u'
add(2, y=5, 10) # is not possible', '_i1': u'n = 5\nf = 1\nfor x i
n range(1, n+1):\n     f = f * x\nprint(f)', '_i44': u'res = fun(2,
3, k=10)', 'r': 3, 'exit': <IPython.core.autocall.ZMQExitAutocall
object at 0x1034bfa90>, 'get_ipython': <bound method ZMQInteractiv
eShell.get_ipython of <ipykernel.zmqshell.ZMQInteractiveShell obje
ct at 0x103478350>>, '_i28': u'd = dict(a=20, b=30, c=40)\nprint d
', '_i29': u"def submit_form(name, addr, mobile, email):\n    prin
t 'Storing data in the database: {}, {}'.format(name, addr)\n    p
rint 'SMS: {} your application has been submited'.format(mobile)\n
print 'EMAIL: {} your application has been submited'.format(email)
", '_i26': u"add(a=20, b=30, c=40) # ==> add({'a':20, 'b':30, 'c':
40})", '_i27': u'add()', '_i24': u'y = 7\nx = 20 if y < 0 else 30\
nprint x', '_i25': u"def add(**kwargs):\n\n    for var, val in kwa
rgs.items():\n         print var,'->', val", '_i22': u'add(8, 9)',
'_i23': u'add()', '_i20': u'def add(*args):\n    print type(args)\
n    s = 0\n    for x in args:\n        s = s + x\n    return s',
'_i21': u'add(2, 3, 4.0, 5, 6, 7.5, 8, 9)', '_i47': u'res = fun()'
, '_i48': u"g = 9.8\n\ndef fun(n):\n x = 50\n y = 90\n n
= n*10\n    print '---- in side fun()-------'\n    print '----- lo
cals() ----------'\n    print locals()\n    print '----- globals()
----------'\n    print globals()\n\ndef main():\n    x = 20\n    n
= 30\n    fun(n)\n    print '---- in side main()-------'\n    prin
t '----- locals() ----------'\n print locals()\n print '----
- globals() ----------'\n    print globals()\n\n    \nmain()", '
builtins ': <module ' builtin ' (built-in)>, '_i45': u'res = fu
n(2,3, d=10)', '_i46': u'res = fun(4,5)', '_ih': ['', u'n = 5\nf =
1\nfor x in range(1, n+1):\n    f = f * x\nprint(f)', u"# NCR Prog
ram\n\nn = 5\nr = 2\n\n# n!\nnfact = 1\nfor x in range(1, n+1):\n
nfact = nfact * x\n\n# n-r!\nnrfact = 1\nfor x in range(1, n-r+1):
\n  nrfact = nrfact * x\n    \n# r!    \nrfact = 1\nfor x in ran
ge(1, r+1):\n  rfact = rfact * x\n \n \nprint 'nCr: ', nfa
ct/(nrfact*rfact)", u'def add(x, y): # function definition\n z
= x + y\n return z\n\nresult = add(3, 4) # function call\nprint
result', u"# computing nCr by reusing the factorial program\ndef f
act(n): \n f = 1 \n for x in range(1, n+1): \n f = f
* x\n    return f\n\nn = 5\nr = 3\nres = fact(n)/(fact(n-r)*fact(r
))\nprint 'nCr = ', res", u"# function which doesn't take anything
and doesn't return anything\ndef greet():\n    print 'Hello, How a
re you today?'\n\ngreet()", u'res = greet()\nprint res', u'# defin
ition of function add\ndef add(x, y, z): # formal parameters\n
s = x + y + z\n return s\n\na = 2\nb = 4\n# function call\nfina
l_sum = add(a, 3, b)# actual arguments\nprint final_sum', u'add(4,
5,6)', u'# positional arguments are mandatory arguments, \n# Argum
```

```
ents will be recievd by the formal arguments \n# in the same order
of actual arguments.\n# we cannot skip passing them.\ndef add(x,  y
, z): \n s = x + y + z\n return s\n\nprint add(3, 4, 5) # po
sitional arguments', u'add(2, z=4, y=5) # keyword arguments',  u'ad
```

```
d(2, y=5, 10) # is not possible', u'def add(x, y, z=5): # default
arguments\n    s = x + y + z\n    return s', u"# z takes default v
alue, 5\nprint add(3, 4)\n\n# z's default value replaced by the ac
tucal argument, 10\nprint add(4, 3, 10)", u'def add(x, y=0,  z=0):
# deafult arguments\n    s = x + y + z\n    return s\n\nprint add(
4)', u'add(4, 5)', u'print add(4, z=5)', u'def add(x=0, y, z=0): \
n    s = x + y + z\n    return s\n\nprint add(4, 5)', u'def add(x=
0, y=0, z=0): # deafult arguments\n s = x + y + z\n return  s
\nprint add()', u'print add(z=5, y=4)# if we want pass value to  z'
, u'def add(*args):\n    print type(args)\n    s = 0\n    for x in
args:\n         s = s + x\n    return s', u'add(2, 3, 4.0, 5, 6, 7.
5, 8, 9)', u'add(8, 9)', u'add()', u'y = 7\nx = 20 if y < 0 else 3
0\nprint  x', u"def  add(**kwargs):\n\n for  var,  val  in  kwargs.it
ems():\n print var,'->', val", u"add(a=20, b=30, c=40) # ==
> add({'a':20, 'b':30, 'c':40})", u'add()', u'd = dict(a=20, b=30,
c=40)\nprint d', u"def submit_form(name, addr, mobile, email):\n
print 'Storing data in the database: {}, {}'.format(name, addr)\n
print 'SMS: {} your application has been submited'.format(mobile)\
n    print 'EMAIL: {} your application has been submited'.format(e
mail)", u"submit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.com')", u'
def submit_form(name, addr, citizen=\'IND\', *args, **kwargs):\n
"""\nsyntax  :  submit_form(name,  addr,  nationality=\'Indian\')\
n    param name(mandatory): str name of  the candidate\n    param a
ddr(optional): str address\n    param nationality(deafult=\'IND\')
: str nationality\n    param mobile(optional): str valid phone \n
param email(optional): str valid email\n    """\nprint    kwargs\
n    print \'Storing data in the database: {}, {}, {}\'.format(nam
e, addr, citizen)\n    \n    if \'mobile\' in kwargs:\n         pri
nt \'SMS: {} your application has been submited\'.format(kwargs[\'
mobile\'])\n    if \'email\' in kwargs:\n        print \'EMAIL: {}
your application has been submited\'.format(kwargs[\'email\'])\n
', u"submit_form('Ashok', 'hyderabad', mobile='9875976554')", u"su
bmit_form('jhon', 'Delhi', email='jhon@gmail.com')", u"submit_form
('Karthik', \n                  'Chennai', \n                    m
obile='9875976554', \n                  email='karthik@gmail.com'
)", u"submit_form('Naren', 'Hyd')", u"def simple_interest(principl
e, **kwargs):\n\n    duration = kwargs.get('duration', 12)\n    ra
te = kwargs.get('rate', 0.12)\n\n    return principle*duration*rat
e ", u'simple_interest(2100000, duration=24)', u'add(x=2, y=3, z=4
0)', u"d = {'x': 20, 'y':30, 'z':40}\nprint d", u'd = dict(x=20,  y
=30, z=40)\nprint d', u"def fun(a, b, c=10, d=20, *args,  **kwargs)
:\n    s = a + b + c + d\n    print '-------------------'\n    pri
nt 'Positional arguments'\n    print '-------------------'\n    pr
int 'a = ', a, ' b = ', b\n    \n    print '-------------------'\n
print 'Default arguments'\n    print '-------------------'\n    pr
int 'c = ', c, ' d = ', d\n    \n    print '-------------------'\n
print 'Variable arguments'\n    print '-------------------'\n    p
rint args\n\n    print '-------------------'\n    print 'Variable
Keyword arguments'\n    print '-------------------' \n    print kw
args\n    \n    return s", u'res = fun(2, 3, 4, 5, 6, 7, 8, p=10,
q=20)', u'res = fun(2, 3, 5, 6)', u'res = fun(2,3, k=10)', u'res  =
fun(2,3, d=10)', u'res = fun(4,5)', u'res = fun()', u"g = 9.8\n\nd
ef fun(n):\n    x = 50\n    y = 90\n    n = n*10\n    print '----
in side fun()-------'\n    print '----- locals() ----------'\n
```

```
print locals()\n    print '----- globals() ----------'\n    print
globals()\n\ndef main():\n    x = 20\n    n = 30\n    fun(n)\n
print '---- in side main()-------'\n    print '----- locals() ----
------'\n    print locals()\n    print '----- globals() ----------
'\n   print globals()\n\n   \nmain()"], '_i40': u'd = dict(x=20,
y=30, z=40)\nprint d', '_i41': u"def fun(a, b, c=10, d=20,  *args,
**kwargs):\n    s = a + b + c + d\n    print '-------------------'
\n    print 'Positional arguments'\n    print '-------------------
'\n    print 'a = ', a, ' b = ', b\n    \n    print '-------------
------'\n    print 'Default arguments'\n    print '---------------
----'\n    print 'c = ', c, ' d = ', d\n    \n    print '---------
----------'\n    print 'Variable arguments'\n    print '----------
---------'\n    print args\n\n    print '-------------------'\n
print 'Variable Keyword arguments'\n    print '-------------------
' \n    print kwargs\n    \n    return s", '_i42': u'res = fun(2,
3, 4, 5, 6, 7, 8, p=10, q=20)', '_i43': u'res = fun(2, 3, 5,  6)',
'y': 7, '_10': 11, '__name__': '__main__', '___': 17, '_': 6048000
.0, 'a': 2, 'g': 9.8, 'greet': <function greet at 0x1035a00c8>, '_
i39': u"d = {'x': 20, 'y':30, 'z':40}\nprint d", '_i38':  u'add(x=2
, y=3, z=40)', '_37': 6048000.0, '_ii': u'res = fun(4,5)',  '_i6':
u'res = greet()\nprint res', '_i31': u'def submit_form(name, addr,
citizen=\'IND\',  *args,  **kwargs):\n   """\n  syntax  :  submit_f
orm(name, addr, nationality=\'Indian\')\n     param name(mandatory)
: str name of the candidate\n    param addr(optional): str  address
\n    param nationality(deafult=\'IND\'): str nationality\n    par
am mobile(optional): str valid phone \n    param email(optional):
str valid email\n    """\n    print kwargs\n    print \'Storing da
ta in the database: {}, {}, {}\'.format(name, addr,  citizen)\n
\n    if \'mobile\' in kwargs:\n       print \'SMS: {} your appli
cation  has  been  submited\'.format(kwargs[\'mobile\'])\n  if  \'em
ail\' in kwargs:\n        print \'EMAIL: {} your application has b
een   submited\'.format(kwargs[\'email\'])\n   ',   '_i30':   u"sub
mit_form('Ashok', 'Hyd', 0, 'ashok@hotmail.com')", '_i33': u"submi
t_form('jhon', 'Delhi', email='jhon@gmail.com')", '_i32':  u"submit
_form('Ashok', 'hyderabad', mobile='9875976554')", '_i35': u"submi
t_form('Naren', 'Hyd')", '_i34': u"submit_form('Karthik', \n
'Chennai', \n                          mobile='9875976554', \n
email='karthik@gmail.com')", '_i37': u'simple_interest(2100000, du
ration=24)', '_i36': u"def simple_interest(principle,  **kwargs):\n
\n    duration = kwargs.get('duration', 12)\n    rate = kwargs.get
('rate', 0.12)\n\n    return principle*duration*rate  "}
```

Below is the output of above program:

```
---- in side  fun()-------

----- locals() ----------
{'y': 90, 'x': 50, 'n': 300}
----- globals() ----------
{'g': 9.8, '__builtins__': <module '__builtin__' (built-in)>, '__file_
': '/Users/Nikky/Project/local_global.py', '_package_': None,  'fun':
<function fun at 0x100495f50>, '__name__': '__main__', 'main': <functio
n main at 0x10049b050>, '_doc_': None}

---- in side main()-------

----- locals() ----------
{'x': 20, 'n': 30}
----- globals() ----------
{'g': 9.8, '__builtins__': <module '__builtin__' (built-in)>, '__file_
': '/Users/Nikky/Project/local_global.py', '_package_': None,  'fun':
<function fun at 0x100495f50>, '__name__': '__main__', 'main': <functio
n main at 0x10049b050>, '_doc_': None}
```

locals(): This function returns all the local identifiers(varibles and any functions) of that function.

globals(): This function returns all the global identifiers(varibles and any functions) whicha are available outside.

g is a global variable in the above program which is available in all the functions. And each function is availble to all other functions including to itslef.

global keyword

```
In[]  g = 9.8
      def fun():
          g = 10

      def start():
          print g
          fun()
          print g

      start()
```

```
9.8
9.8
```

```
In[]  g = 9.8

      def fun():
          global g
          g = 10

      def start():
          print g
          fun()
          print g

      start()
```

```
9.8
10
```

Understanding function Call stack

In the below example each function has its own set of local veraibles. The place where these locals are stored is called function call stack of that function. Stack frame of a function gets destroyed before control leaving the function. A function is alive as long as its stack frame resides in memory. The memory layout where all the stack frames are created is called function call stack.

In[]
```
g = 9.8

def fun3(x):
    print 'fun3 start'
    print 'stack frame of fun3: ', locals()
    print 'fun3 end'

def fun2(n):
    print 'fun2 start'
    y = 30
    n = n + 1
    print 'stack frame of fun2: ', locals()
    fun3(n)
    print 'fun2 end'

def fun1(n):
    print 'fun1 start'
    n = n + 1
    x = 20
    print 'stack frame of fun1: ', locals()
    fun2(n)
    print 'fun1 end'

def main():
    print 'Main starts here'
    a = 100
    print 'stack frame of main: ', locals()
    fun1(a)
    print 'Main ends here'

if_name == '_main_':
    main()
```

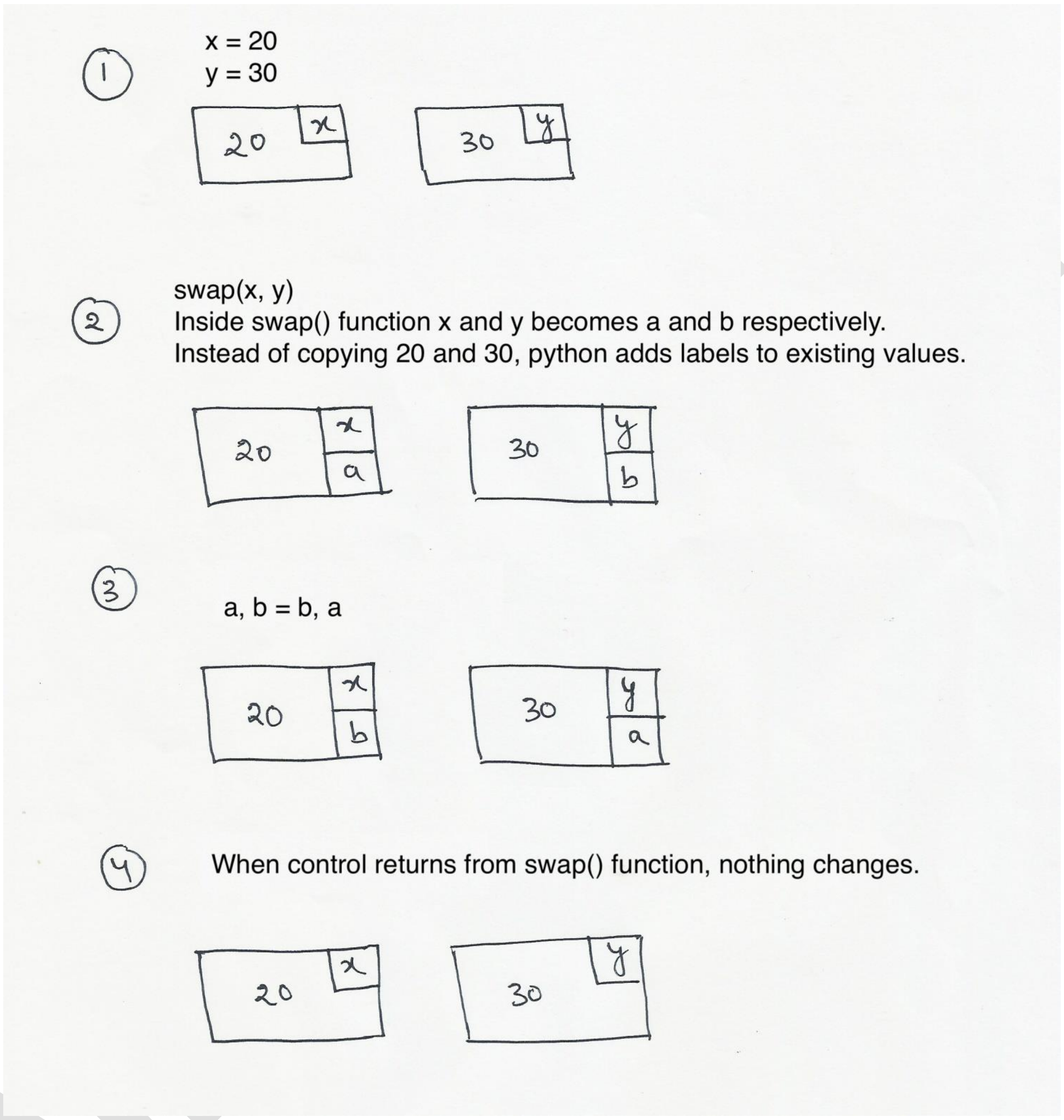
```
Main starts here
stack frame of main:  {'a': 100}
fun1 start
stack frame of fun1:  {'x': 20, 'n': 101}
fun2 start
stack frame of fun2:  {'y': 30, 'n': 102}
fun3 start
stack frame of fun3:  {'x': 102}
fun3 end
fun2 end
fun1 end
Main ends here
```

call-by-object-reference

In[]
```
def swap(a, b):
    a, b = b, a

x = 20
y = 30
swap(x, y)

print 'x = ', x, ' y = ', y
```

```
x =  20  y =  30
```

In the above example, 'x' and 'y' values are not changed as 'a' an 'b' will be new lables for 20 and 30, infact we are only exchanging lables for them. As all the primitive types are immutable, there is no affect on 'x' and 'y'.



① x = 20
y = 30

② swap(x, y)
Inside swap() function x and y becomes a and b respectively.
Instead of copying 20 and 30, python adds labels to existing values.

③ a, b = b, a

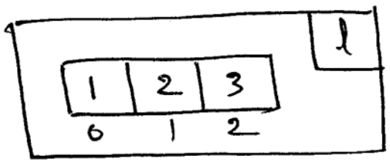
④ When control returns from swap() function, nothing changes.

In[]
```
# Arguments are sent to a function by  call-by-object-reference

def modify(p):
    p[1] = 555

def main():
    l = [1, 2, 3]
    modify(l)
    print l

if_name == '_main_':
    main()
```

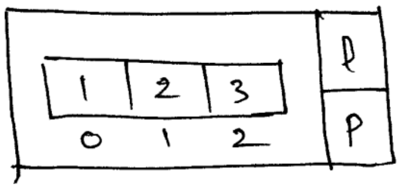
```
[1, 555, 3]
```

In the above example p is another label for same list(i.e, l) and we are accessing individual element of list l through p. So once we come back from modify() function there will be effect on l as l and p are referring same list.

1 l = [1, 2, 3]

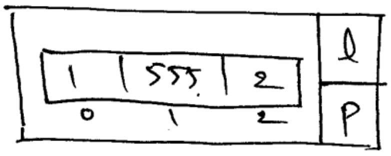


2 modify(l)

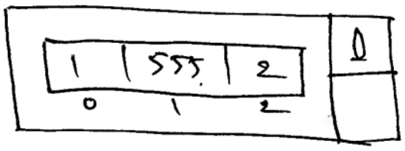


3 inside modify() ‘l’ is ‘p’

p[1] = 555



4 After returning from modify() function



In[]
```
# replacing with with another list

def modify(p):
    p = [4, 5, 6]

def main():
    l = [1, 2, 3]
    modify(l)
    print l

if_name == '_main_':
    main()
```

```
[1, 2, 3]
```

if we completely replace p with some other value(it can be a list or single value), there is no effect on l, afterall p is just an another lable for same list. Label p moves from list [1, 2, 3] to [4, 5, 6].

How do we prevent modifying 'l' in function modify ?

Just create a copy and pass it to modify() function

copy - shallow copy

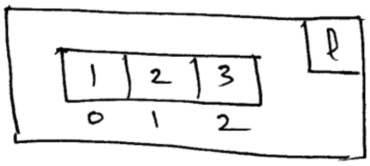
When we want to create duplicate copy of the elements in a container(list, set, dictionary etc) we use copy function from copy module.

In[]
```
import copy

def modify(p):
    p[1] = 555

def main():
    l = [1, 2, 3]
    dup = copy.copy(l)  # Alternative l.copy()
    modify(dup)
    print l

if_name == '_main_':
    main()
```

```
[1, 2, 3]
```

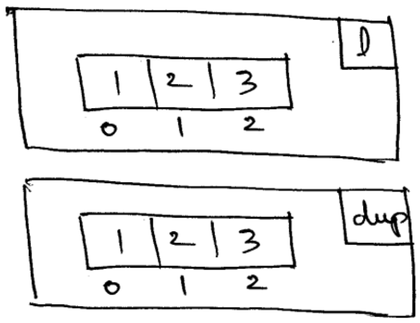
1

l = [1, 2, 3]



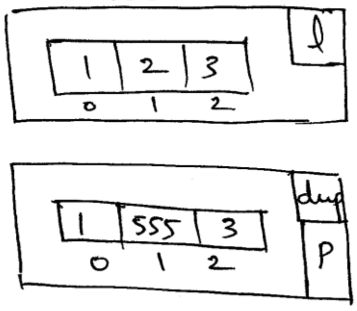
2

dup = copy.copy(l)



3

Inside modify function
p[1] = 555

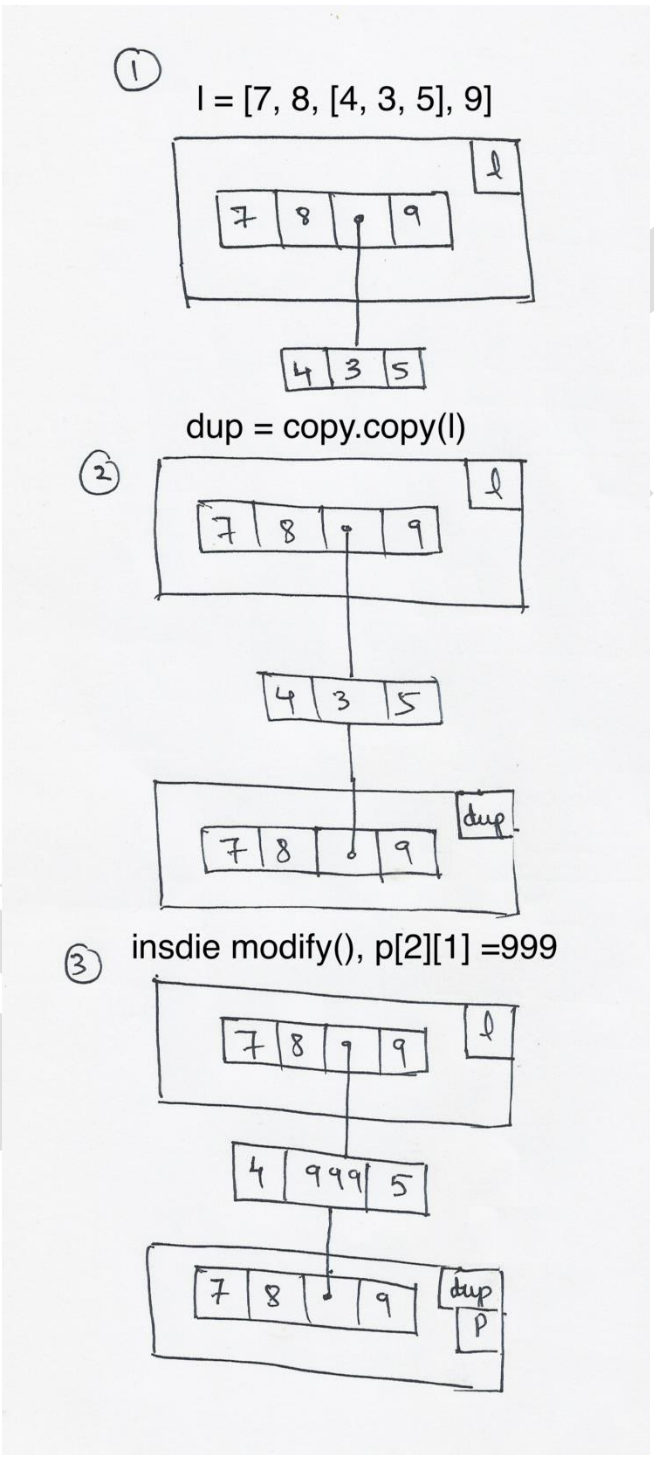


In[]
```
import copy

def modify(p):
    p[2][1] = 999

def main():
    l = [7, 8, [4, 3, 5], 9]
    dup = copy.copy(l)
    modify(dup)
    print l

if_name == '_main_':
    main()
```

```
[7, 8, [4, 999, 5], 9]
```

Why values in inner list are being modified?

Because copy() copies elements in the first level only. We have a solution for it.



deepcopy

copy() function only copies elements in the first level, but deepcopy copies all the objects even

they are in multiple levels. e.g, list of lists, list of dictionaries etc.

In[]
```
import copy

def modify(p):
    p[2][1] = 999

def main():
    l = [7, 8, [4, 3, 5], 9]
    dup = copy.deepcopy(l)
    modify(dup)
    print l

if_name == '_main_':
    main()
```
```
[7, 8, [4, 3, 5], 9]
```

Recursion

"A function calling itself is called recursion."

Recursion is generally used when a problem has recursive sub problems.

In [ ]:
```
def fun():
    print 'Apple'
    fun()

fun()
```

Above function repeatedly prints 'Apple', and never stops. If we use the recursion with control, we can solve complex problems easily. A condition which controls recursion is called base-case.

In[]
```
def fun(n):
    if n > 0: # base-case
        print n
        fun(n-1)
fun(5)
```
```
5
4
3
2
1
```

Program: Calculating factorial using recursion

```
In[] def fact(n):
         if n == 0 or n == 1:
             return 1
         return n * fact(n-1)

     fact(4)
```

```
Output: 24
```

Program: Flatten the below list
l = [34,5, [4, 5, 7, [7, 19, 9, 1, 2], 5], 13, 4, [6, 14]]

```
In[] l = [34,5, [4, 5, 7, [7, 19, 9, 1, 2], 5], 13, 4, [6,  14]]
     fl = []
     def flatten(l):
         for x in l:
             if type(x) is list:
                 flatten(x)
             else:
                 fl.append(x)

     flatten(l)
     print 'List:', l
     print 'Flattened List', fl
```

```
List: [34, 5, [4, 5, 7, [7, 19, 9, 1, 2], 5], 13, 4, [6,  14]]
Flattened List [34, 5, 4, 5, 7, 7, 19, 9, 1, 2, 5, 13, 4, 6,  14]
```

Passing a function to another function

Functions are first class objects in python. We can pass a function to another function like any other type.

```
In[] def fun():
         print 'Hello'

     print fun
```

```
<function fun at 0x103db48c0>
```

In the below example greet method is passed to 'fun' function.

In[]
```
def greet():
    print "Welcome Pythonians!!!"

def fun(f):
    # ------
    f()

    # -----
fun(greet)
```

```
Welcome Pythonians!!!
```

Defining a function within a function

In[]
```
def fun():
    pi = 22/7.0
    def area(r):
        a = pi* r*r
        print "Area = ", a

    print "-------"
    area(8)
    print "-------"

fun()
```

```
-------
Area =  201.142857143
-------
```

area() is a function which is defined inside fun() funciion asn can only be accessible to fun().

Returning a function from a function

In[]
```
def fun():
    pi = 22/7.0
    def area(r):
        a = pi* r*r
        print "Area = ", a
    return area

x = fun()
x(4)
```

```
Area =  50.2857142857
```

In[]
```
x(4)
```
```
Area =  50.2857142857
```
In[]
```
fun()(4)
```
```
Area =  50.2857142857
```

Passing a function to another function along with its arguments

In[]
```
def add(x, y):
    print x + y

def volume(h, l, b):
    print h*l*b

def compute(callback, *args):
    # some code here
    callback(*args)
    # some code here

compute(add, 2,3)
compute(volume, 4, 5, 6)
```
```
5
120
```

Decorator

Decorator is a design pattern, which is used to additional functionality to a function dynamically. In the below example stars function is adding addtional stars to the ouput of every function that is being passed.

```
In[]  def stars(f, *args, **kwargs):
          print "***********"
          f(*args, **kwargs)
          print "***********"

      def add(x, y):
          print x + y

      def greet():
          print "Hello World!"

      stars(add, 3, 4)
      stars(greet)
```

```
***********
7
***********
***********
Hello World!
***********
```

Python provides smart and cleaner syntax to achieve this,

```
In[]  def stars(f):
          def wrapper(*args, **kwargs):
              print "***********"
              ret = f(*args, **kwargs)
              print "***********"
              return ret
          return wrapper


      @stars
      def add(x, y):
          s = x + y
          print s

      @stars
      def greet():
          print "Hello World!"

      add(3, 4)
      greet()
```

```
***********
7
***********
***********
Hello World!
```

****

Whenever we want to call the function with additional functionality, We don't need to pass the function to stars(), instead, just add @stars on the top of function definition, that adds additional functionality to function definition iteself.

Practical use-case

In[]
```
import time

def calc_time(func):
    def time_wrapper(*args, **kwargs):
        start = time.clock()
        ret = func(*args, **kwargs)
        end = time.clock()
        print "Time taken: ", end - start
        return ret
    return time_wrapper

@calc_time
def process(n):
    for i in xrange(n):
        i = i * i

process(10000000)
```

```
Time taken:  0.54261
```

Closure

Closure is the context captured by an inner function, which will be used out-side the outer function scope, even parent is not alive.

In[]
```
def n_circles(n):
    global y
    pi = 22/7.0

    def area(r):
        print 'Local vars of area:', locals()
        a = pi*r*r*n
        return a

    return area


x = n_circles(5)
print x(3)

print x.func_closure[0].cell_contents
print  x.func_closure[1].cell_contents
```

```
Local vars of area: {'pi': 3.142857142857143, 'r': 3, 'n': 5}
141.428571429
5
3.14285714286
```

In the above example , we are returning area() function from n_circles() functions and assigned to x. Here 'x' is 'area'. We can call x() now. If we look at the out put, we can see 'n' and 'pi' as local variables of 'x'. Infact 'x' is function area(). 'n' and 'pi' are local variables of n_circles() not area(). But how area() is able to use them even after n_circles() exit. This is called 'closure'. Function area() captured all varibales required for its execution. In python, functions are objects. This function object has a property called 'func_closure', a tuple of captured values. In the above out put we can see how to access the content of closure.

Generator

"A function with 'yield' statment instead 'return' is called as generator in python." Instead of calling entire function each time. Python creates a context for function with yield statment and returns a generator object.

- **Generator returns a value by executing next iteration of the generator expression.**
- **we can pass a generator to 'next()' function, to get the subsequent value of the generator.**

Note:

- **iterators are used to traverse exisiting data**
- **Generators produces values on demand and also can be used as iterators**

Creating Custom generators

```
In[] def fun(n):
         i = 1
         while i <= n:
             return i
             i += 1

     fun(5)
```

```
Output: 1
```

Above function call fun(5), always returns 1, as loop ends in the fist iteration iteself.

Replace 'return' wth 'yield'

```
In[] def fun(n):
         i = 1
         while i <= n:
             yield i
             i += 1

     gen = fun(5)
     print type(gen)
```

```
<type 'generator'>
```

now, fun(5) returns a genrator. We can get next value from a generator using built-in next() function

```
In[] next(gen)
```

```
Output: 1
```

next value,

```
In[] next(gen)
```

```
Output: 2
```

and so on ...

We can also is this generator in for loop.

In[]
```
for i in fun(5):
    print i
```

```
1
2
3
4
5
```

After values exhausted, next returns StopIteration error.

In[]
```
next(gen)
```

Output:
```
3
```

to prevent this we can have a default values after values exhausted.

In[]
```
print next(gen, None)
```

```
4
```

Program: Implement xrange() function

In[]
```
def my_xrange(*args):
    if len(args) <=3 and len(args) >= 1:
        start = 0
        step =1

        if len(args) == 1:
            end = args[0]

        elif len(args) == 2:
            start = args[0]
            end = args[1]

        elif len(args) == 3:
            start = args[0]
            end = args[1]
            step = args[2]

        i = start
        while i < end:
            yield i
            i += step
    else:
        print 'Invalid parameter count'
```

```
In[]  for x in my_xrange(1, 11, 3):
          print x
```

```
1
4
7
10
```

Explicit iterators

We cannot resume the exection once we break the iteration of any sequence. Because we do not have the control on implicit itertor object maintained by for loop.

```
In[]  l = [34, 40, 32, 31, 35, 33, 37]

      s = 0
      for t in l:
          if s + t > 150:
              break
          print t
          s += t

      print 'Remaining values:'

      for t in l:
          print t
```

```
34
40
32
31
Remaining values:
34
40
32
31
35
33
37
```

We can use iter() function to create on iterator object, which can be controlled by the developer. If we use this iterator object to iterate a sequence, we can resume the iteration, even after breaking the iteration abruptly.

using iter()

In[]
```
l = [34, 40, 32, 31, 35, 33, 37]
it = iter(l)

s = 0
t = 0
for t in it:
    if s + t > 150:
        break
    print t
    s += t

print 'Remaining values:'
print t
for t in it:
    print t
```

```
34
40
32
31
Remaining values:
35
33
37
```

Interview questions

1. How do you write custom generators?
2. What does the yield statement do?
3. What is Recursion? Give Example?
4. What is decorator, usage?
5. Write a function decorator in Python?
6. How are arguments passed by value or by reference in python?
7. Mention what are the rules for local and global variables in Python?
8. How to use *args and **kwargs in python?
9. What are positional arguments?
10. What are default arguments?
11. What are keyword arguments?
12. What are variable arguments?
13. What are variable keyword arguments?
14. What is a closer in Python?
15. When do you use variable keyword arguments?
16. How can you get all global variables and local variables in a scope?
17. How "call by value" and "call by reference" works in Python?
18. Difference between "cmp()" function, "==" and "is"?
19. Mention the use of the split() function in Python?
20. What is the purpose of zip(), enumerate()? How to unzip list of tuples to multiple lists?
21. How can you implement functional programming and why would you?
22. What is lambda and how it works in Python?
23. How method overloading works in Python?
24. Difference between "deepcopy" and "shallow copy"
25. What is docstring in Python?
26. What is pure function?
27. What is the use of next function?
28. Program: Flatten the below list using recursion

```
l= [34, 5, [ 4, 5, 3, [ 3, 19, 9, 1, 2 ] , 5], 13, 4, [ 6, 14]  ]
```

```
In [ ]:
```