Simple Function

```
def hello():
    print("Hello, how are you doing today?")
hello()
```

Function with an argument

```
def hello(name):
    print("Hello {}, how are you doing today?".format(name))
hello("Sandya")
hello("Ram")
```

Function with two arguments

```
def hello(name,age):
    print("{} is {} years old".format(name,age))
hello("Sandya",10)
hello("Ram",9)
```

Exercise on Function

Write a function diff which takes two parameters and returns their difference.

Ex:

diff(5,2) should return 3

diff(2,5) should return 3

Do not use abs() inside your function

Exercise

```
def diff(x,y):
 if x>y:
    return x-y
 else:
    return y-x
```

Function with arguments with default values

```
def add(x,y=10):
    return x+y
c=add(4,5)
d=add(5)
print(c,d)
print(add(5,6),add(8))
```

Keyword arguments

```
def wish(name,age):
    print("Hello {} you are {} years  old".
format(name,age))


wish('India',71)
wish(71,'India')

wish(age=71,name='India')
```

global

```
age=56
def grow():
    print(age)
grow()
```

global

```
age=56
def grow():
    print(age)
    age=age+1
    print(age)
grow()
 print(age)
#Note: local variable 'age' referenced before assignment
```

global

```
age=56
def grow():
    global age
    print(age)
    age=age+1
    print(age)
grow()
 print(age)
```

Function that returns multiple values

```
#Python function can return any number of values
def sumdiff(a,b):
   return a+b,abs(a-b)
print(type(sumdiff(4,9)))
mysum,mydiff=sumdiff(4,9)
print(mysum,mydiff)
```

Handling variable length values

```
def average(*num):
   print(type(num))
    print(num)
    print(float(sum(num))/len(num))
average(3,4)
average(3,4,8)
average(3,4,8,90,4.5,5.3,7.8)
average()
#*args will pack all the arguments into a tuple
#called args
```

Handling variable length arguments

```
def average(a,b,*num):
   print(type(num))
    print(num)
    print((a+b+sum(num))/(2+len(num)))
average(3,4)
average(3,4,8)
average(3,4,8,90,4.5,5.3,7.8)
average()
#*args will pack all the arguments into a tuple
#called args
```

Variable length keyword arguments

```
def polygon(**kwds):
    print(type(kwds))
    print(kwds)
polygon(width=10,length=20)
polygon(width=10,length=20,height=5)
polygon(width=10,length=20,height=5,units='cm')


#**kwds will pack all the arguments into a dict
#called kwds
```

Handling any type of arguments

```
def polygon(a,b,c,*sides,**options):
   print(type(options))
   print(type(sides))
   print(a,b,c)

polygon(8,7,6,4,2,8,units='cm',compute='area')

#**kwds will pack all the arguments into a dict
#called kwds
```

Lets revisit functions

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"
```

Which call is correct?

- parrot(1000)
- correct
- parrot()
- wrong - required argument missing
- parrot(action = 'VOOOOOM', voltage = 1000000)
- correct
- parrot('a thousand', state = 'pushing up the daisies')
- correct

Lets revisit functions

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print "-- This parrot wouldn't", action,
    print "if you put", voltage, "volts through it."
    print "-- Lovely plumage, the", type
    print "-- It's", state, "!"
```

Which call is correct?

- parrot('a million', 'bereft of life', 'jump')
- correct
- parrot(110, voltage=220)
- wrong - duplicate value for argument
- parrot(voltage=5.0, 'dead')
- wrong - non-keyword argument following keyword