Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

A function can be Pre-defined or User-defined.

A function can have parameters.

```
# Functions in python
# A task within a named block
# Used to execute a block of code for any no of times
# Code reusability
# To create a fun in python we use the keyword 'def'
# Functions support if-else conditions and loops
# A fun name cannot be a keyword
# A fun call should always be after fun definition
# Do not write spaces, symbols or start with numbers while giving a fun name

'''
# defining a fun
def add():
    i,j = 11,22
    print("Add of i and j is: ", (i+j))
    pass

print("This is main area")
add() # fun call
print("End of the prog")
'''

'''
# A single python file can have any no of functions
# Make sure all the fun names are distinct(different)

def add():
    i,j = 11,22
    print("Add of i and j is: ", (i+j))
    pass


def sub():
    i,j = 11,22
    print("Sub of i and j is: ", (j-i))
    pass
```

```
def mul():
    i,j = 11,22
    print("Mul of i and j is: ", (i*j))
    pass

def div():
    i,j = 11,22
    print("Div of i and j is: ", (j/i))
    pass

# You can call a fun in any order

mul()
add()
div()
sub()
'''

'''
# Fun calling with duplicate names
# Always the last duplicate fun will override the previous one
def add():
    i,j = 11,22
    print("Add of i and j is: ", (i+j))
    pass

add()

def add():
    i,j = 1,2
    print("Add of i and j is: ", (i+j))
    pass

add()
'''

'''
# fun calling in a loop

def add():
    i,j = 11,22
    print("Add of i and j is: ", (i+j))
    pass
```

```
for i in range(1,6):
    add()
    pass

print("************")
i = 1
while(i<=5):
    add()
    i=i+1
    pass

'''

'''
# loop in a funtion
def generateNumbers():
    print("Fun starts")
    for i in range(1,11):
        print("value is: ", i)
        pass
    print("Fun ends")
    pass


generateNumbers()
'''

'''
# calling a fun in an if or else condition
def generateNumbers():
    print("Fun starts")
    for i in range(1,11):
        print("value is: ", i)
        pass
    print("Fun ends")
    pass


a = int(input("Enter a number: "))
if (a>0 and a<=100):
    generateNumbers()
else:
    print("Pls provide numbers from 1 to 10")
'''
```

```
'''
def generateNumbers():
    print("Fun starts")
    a = int(input("Enter a number: "))
    if (a>=0 and a<=100):
        for i in range(1,a+1):
            print(i, end=' ')
            pass
        pass
    print("Fun ends")
    pass

for i in range(1,6):
    generateNumbers()


'''


# Local and Global Variables
# Global var's can be accessed throught any fun in the python file
# Local var's can only be accessed inside that fun by default.

i = 10 # global var

def sample():
    j = 20 # local var
    print(i)
    print(j)
    pass

#sample()
print(i)
print(j) # NameError
```

Function ‘return’ type and arguments

```
# return value
# we use 'return' keyword to get a local variable to the calling fun
# or main area
# return sholud be used only within function.
# it is recommended to write the 'return' stmt only as the last line of the fun.
# 'return' also get the control out of the fun.
def test():
    i = 20  #local var
    return i


i = test()
print("From main: ", i)
'''

'''



# Function override
# When two fun's have the same name but with different body, then python
# will override the first fun with the second fun.
# If we want to execute(call) the first fun, then it should be called
# before defining the second fun.
def one():
    i, j = 2, 3
    print('From fun one: ', i + j)
    pass


one()


def two():
    i, j = 2, 3
    print('From fun one: ', i - j)
    pass


one()
one()
one()
```

```
'''

# Funtion parameters/arguments

'''


# function with no parameters
def one():
    i, j = 2, 3
    print('From fun one: ', i + j)
    pass


# fun with single parameter
def two(i):
    j = 3
    print("From fun two: ", i + j)
    pass


# fun with two or multi parameters
def three(i, j):
    print("From fun three: ", i + j)
    pass


one()
two(6)
three(4, 15)
'''

'''


def getDetails(id, name, age, gender):
    print("Id is: ", id)
    print("Your name is: ", name)
    print("Your age is: ", age)
    print("Your gender is: ", gender)
    pass


#getDetails(101,'tajesh',20,'M')
print("Enter your id,name,age and gender")
id = int(input())
```

```
name = str(input())
age = int(input())
gender = str(input())

print("\nYour details are: \n")
getDetails(id, name, age, gender)
'''

'''
# Function with both return value and paramerters


def getDetails(id, name, age, gender):
    id = id + 10
    name = 'Mr. ' + name
    age += 5  # (or) age=age+5
    return id, name, age, gender


print("Enter your id,name,age and gender")
id = int(input())
name = str(input())
age = int(input())
gender = str(input())

print("\nYour details are: \n")
details = getDetails(id, name, age, gender)
print("Details in a tuple: ", details)

i, n, a, g = getDetails(id, name, age, gender)
print("your details are: ", i, n, a, g)
'''

'''
# Recursion:
# It is a process of calling a funtion itself


def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
```

```
        return (x * factorial(x - 1))


num = int(input("Enter a number to get its factorial: "))
print("The factorial of", num, "is", factorial(num))
'''


# fun calling another fun
def one():
    print("This is fun one")
    pass

def two():
    one()
    print("This is fun two")
    pass

def three():
    two()
    print("This is fun three...")
    pass

# main area
three()
'''
```

```
'''
Default Arguments
A default argument is an argument that assumes a default value
if a value is not provided in the function call for that argument.
The following example gives an idea on default arguments,
it prints default age if it is not passed.
'''

# Function definition is here
def printinfo( name, age = 35 ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return

# Now you can call printinfo function
printinfo( age = 50, name = "RamaRao" )
printinfo( name = "Shankar" )
```

```
'''
Variable-length Arguments:

You may need to process a function for more arguments than you specified while
defining the function. These arguments are called variable-length arguments and
are not named in the function definition, unlike required and default arguments.

An asterisk (*) is placed before the variable name that holds the values of all
nonkeyword variable arguments. This tuple remains empty if no additional
arguments are specified during the function call.
'''

# Function definition is here
def printinfo( arg1, *params ):
   "This prints a variable passed arguments"
   print ("Output is: ")
   print (arg1)

   for var in params:
      print (var)
   return

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```