In [1]:

```
#Functions - If a group of statements is repeatedly required then it is not recommended to write these statements everytime
# seperately. We have to define these statements as a single unit and we can call that unit any no of times based on our
# requirement without rewriting. This unit is nothing but function.

# The main advantage of function is code reusability.
# In other languages functions are also known as methods, procedures, subroutines etc;
```

In [2]:

```
# python supports 2 types of functions
# 1.built in functions
# 2.user-defined functions.
```

In [3]:

```
# 1.built in functions => The functions which are coming along with python software automatically are called built in
# functions or pre defined functions

#eg:
# id()
# type()
# input()
# eval()
```

In [4]:

```
# user defined functions - the functions which are developed by programmers explicitly according to business requirements,
# are called user defined functions.

#Syntax to create user defined functions:
# def functionname(parameters):
#  '''doc string'''
# return value
```

In [5]:

```
# while creating a function we can use 2 keywords
# 1.def(mandatory)
# 2.return (optional)
```

In [6]:

```
# write a function to say hi
def wish():
    print("hi")
wish()
wish()
```

```
hi
hi
```

In [7]:

```
# parameters - parameters are inputs to function. If a function contains parameters, then at the time of calling,
# compulsory we should prvoide values otherwise we will get erro
```

In [18]:

```
#eg:
def emp(name,no):
    print('hi',name,'welcome to office','and your id no is',no)
emp('aravind',1)
emp('rahul',2)
```

```
hi aravind welcome to office and your id no is 1
hi rahul welcome to office and your id no is 2
```

In [19]:

```
# Return statement - function can take input values as parameters and executes business logic, and returns output to the
# caller with return statment
```

In [31]:

```
def emp(name,salary,cutoff):
    pay=salary-cutoff
    return pay
print(emp('aravind',85000,1000))
print(emp('kushal',60000,1000))
```

```
(84000, 'aravind')
(59000, 'kushal')
```

In [35]:

```
# returning multiple values from function
def emp(name,salary,cutoff):
    pay=salary-cutoff
    return pay,name
print(emp('aravind',85000,1000))
print(emp('kushal',60000,1000))

#note: In other languages like c,c++ and java, function can return atmost one value, But in python a function can
return
# any no of values
```

```
(84000, 'aravind')
(59000, 'kushal')
```

In [36]:

```
# Types of arguments
# def f1(a,b):
# -----
# -----
#f1(10,20)
# Here a,b are formal arguments where as 10,20 are actual arguments.
# There are 4 types of actual arguments allowes in python.
# 1.positional arguments.
# 2.keyword arguments.
# 3.default arguments.
# 4.variable length arguments.
```

In [39]:

```
# positional arguments => these are the arguments passed to function in correct positional order
def emp(name,salary):
    return name,salary
print(emp('aravind',85000))

# The no of arguments and position of arguments must be matched. If we change the order then result may be change
d.
# If we change the no of arguments we will get error
```

```
('aravind', 85000)
```

In [42]:

```
# keyword arguments => we can pass argument values by keyword i.e by parameter value.
def emp(name,salary):
    return name,salary
print(emp(name='aravind',salary=85000))
print(emp(salary=50000,name='rahul'))

#Here the order of arguments is not important but no of arguments must be matched.
```

```
('aravind', 85000)
('rahul', 50000)
```

In [47]:

```
# Default arguments => sometimes we can provide default values to our positional arguments
def emp(name,salary=70000):
    return name,salary
print(emp('aravind'))
print(emp('rahul'))
print(emp('karthik',140000))

# if we are not passing any value then default value will be considered

#note : after default arguments we will not take non-default arguments. we will get syntax error
```

```
('aravind', 70000)
('rahul', 70000)
('karthik', 140000)
```

In [48]:

```
# Variable length arguments => some times we will pass variable no of arguments to our function such type of vari
ables
# are called variable length arguments.
# We can declare a variable length argument with * symbol as follows
# Syntax : def f1(*n):
# we can call this function by passing any no of arguments including zero number. internally all these values rep
resented
# in the form of tuple.
```

In [49]:

```
def sum(*n):
    total=0
    for n1 in n:
        total+=n1
    print(total)
sum()
sum(10,20,30)
sum(100,30,200)
```

```
0
60
330
```

In [65]:

```
# we can mix variable length arguments with positional arguments.
def emp(name,*programming_language):
    print(name,end=':')
    for p in programming_language:
        print(p,end=' ')
    print()
emp('aravind','c','c++','java','python')
emp('rahul','c#','java')
```

```
aravind:c c++ java python
rahul:c# java
```

In [69]:

```
# After variable length argument, if we are taking any other arguments then we should provide values as keyword a
rguments.
def emp(*programming_language,salary):
    for p in programming_language:
        print(p,end=" ")
    print(salary)
emp('c','c++',salary=80000)
emp('python','java',salary=74000)
```

```
c c++ 80000
python java 74000
```

In [77]:

```
# we can declare keyword variable length-arguments also for this we have to use **
# Syntax : def f1(**n):
# we can call this function by passing any no of keyword arguments. Internally these keywords will be stored in d
ictionary.

#eg:
def emp(**kwargs):
    for k,v in kwargs.items():
        print(k,"=",v,end=",")
    print()
emp(name='aravind',age=21,salary=85000)
emp(name='rahul',age=23,salary=105000)
```

```
name = aravind,age = 21,salary = 85000,
name = rahul,age = 23,salary = 105000,
```

In [ ]: