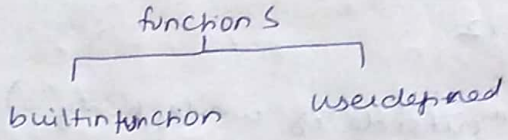
Introduction to Functions: A function is a group of statements that aims to perform a specific task

Advantages of function:

① Avoids repetition ② Reduction in size ③ memory saving ④ code reuse ⑤ Modularisation ⑥ simpler code ⑦ Specific tasks ⑧ parallel development ⑨ Testing ⑩ Portability

Types of functions



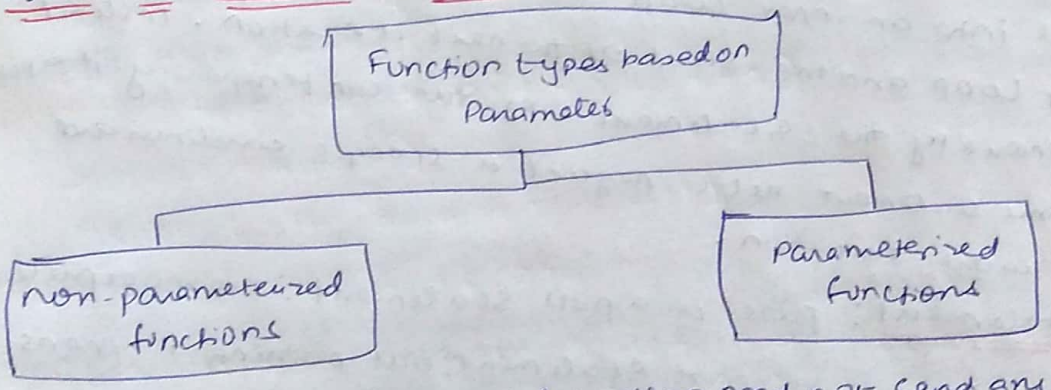
Components of a function are given below:

① function definition
② function call

The function header is the first line of the function. It consists of a keyword def followed by the function name, and an optional list of parameters inside the parentheses. The function header is also known as the function signature.

Caller function → Callee function.

Based on parameters there are basically two types of functions



non-parameterized function: A function need not send any information to the called functions.

```
def warning_message():
    Print("The variable is not used")
warning_message()
```

Parameterised function: A parameterised function passes information as arguments to the functions.

```
def cube(y):
    Print ("The cube is", y* y * y)

cube (10)
```

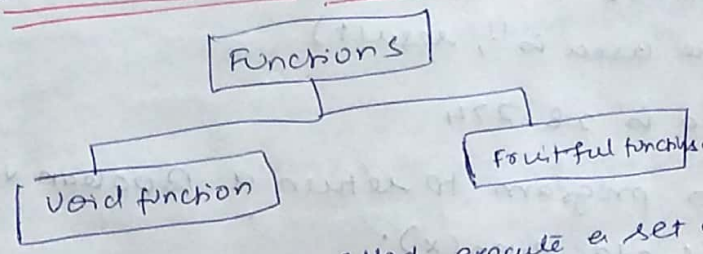
Output: The cube is 100

Passing multiple Arguments

```
def main():
    P= float (input ("Enter the principal amount:"))
    r = float (input ("Enter the rate of interest:"))
    n = float (input ("Enter years of deposit:"))

    Simple_interest = Compute_simple_interest (P,n,r)
    Print ("The simple interest is Rs.", simple_interest)

def Compute_simple_interest (P,n,r):
    return (P*n*r)/100

main()
```

Output: Enter the principle amount: 1000
Enter the rate of interest: 10
Enter the years of deposit: 5
The simple interest is Rs. 500.0

Based on returning value there are two types of functions



The void function, when called, execute a set of instruction functions and terminate. A void function does not return a value.

```
result = print ("Hello world")
result
```

Write a python program to compute some arithmetic operations

```
def Compute(a,b)
    """ This function does basic computation"""
    Print('{0} + {1} = {2}'.format(a,b,a+b))
    Print('{0} - {1} = {2}'.format(a,b,a-b))
    Print('{0} * {1} = {2}'.format(a,b,a*b))
    Print('{0} / {1} = {2}'.format(a,b,a/b))

Compute(100,200)
```

Output:
```
100 + 200 = 300
100 - 200 = -100
100 * 200 = 20000
100 / 200 = 0.5
```

Note: It must be observed that the function call Compute{a,b} makes the assignment a=100 and b=200. Then, in the function, the computations are performed based on the value of a and b. If the order is reversed, say (b,a) then the values would be different. So, the order of the parameter is important

Fruitful function: A function that return a value is called as fruitful function.

```
import math
    def area(radius)
        return(math.pi * radius**2)

    result = area(3)
    Print("The area is", result)
```

Output: The area is 28.274

Write a Python program to return a Boolean value

```
>>> def is_odd_even(x):
        if x%2 = 0:
            return(True)
        else:
            return(false)

is_odd_even(20)
True
is_odd_even(17)
```

write a python program to return multiplication and remainder for the numbers

```
def Cal_mult_div(x,y):
    diff = x/y
    mult = x * y
    return diff, mult

diff, mult = calc_mult_div(10,20)
Print(f"The diff is {diff}, The mult is {mult}")
```

Output: The diff is 0.5, The mult is 200

In this program multiple value are returned by a function. While this is not possible in 'C'

NOTE: valid Return Statement

A function may have one or more return statements. Some of the valid and invalid ~~statement~~ return statements are given below

return, return true, return false, return x, return x,y. and invalid return (x=10, y=10).

Dead code: A function may have multiple return Statements. As soon as the first return statement is encountered, Python immediately ends its execution and the control is returned to the calling program. All codes after the return statement are unreachable. This code is often referred dead code as they are useless and can never be reached.

PROGRAMS FOR DIFFERENT TYPES OF Function based on function arguments.

Function with NO Arguments

```
def print_msg():
    Print("Observe no arguments are passed")

Print_msg()
```

Output: Observe no arguments are passed

Write a program to illustrate positional arguments:

```
def Compute(a,b)
    """ The function does basic computation"""
    Print("sum", format(a+b))
    Print("diff", format(a-b))
    Print("mult", format(a*b))
    Print("div", format(a/b))
    return
Print("Print for parameter (a,b)")
Compute(10,20)
Print("Print for parameter (b,a)")
Compute(20,10)
```

Output:

```
Print for parameters (a,b)
Sum 30
diff -10
mult 200
div 0.5
Print for parameter (b,a)
Sum 30
diff 10
mult 200
div 2.0
```

Note: ① The python interpeter throws error is number of arguments does not match the number of parameters

② Python, when it executes a program creates a special Dunder or magical variable __name__ that stores the name of the program in it. if the program is executed as part of the module, it stores the module name.

Key word Arguments

```
Compute(a=100, b=200)
```

default Argument

```
def compute(a, b=10, c=30):
    Sum = a+b+c
    Print(f"sum is {Sum}")
    return
compute(10)
```

Output: Sum is 50

Variable length Arguments

```
add(10,6)
add(10,7,4)
add(40,50,60,70)
add(50,60,70,80,90)
def function_name(*variable_args):
    ....
    This is a Docstring
    ....
        statements)
        return(expression)
```

Positional arguments

```
def sum(*list)
""" This function finds the sum of variable """
sum = 0
for i in list:
    sum = sum + i
print(sum)
return
sum(10,20)
sum(10,30,40)
sum(1,2,3,4,5)
```

Output: 30, ~~80,5~~
80
15

Keyword variable arguments

```
def sumMarks(**kwargs)
  Print(kwargs)

sumMarks(a=10, b=6)
```

output:

{'a': 10, 'b': 6}

Anonymous Functions: It is also known as lambda

```
import math
distance = lambda x1, y1, y2, y2 : math.sqrt (x1-x2)^2
                                   + (y1-y2)^2)
Print ("distance = {0}", format distance (0,0,15,15)),
```

Output:

```
distance = 03 0.0
```

Note: Call by value

Call by reference

local variable

Global variable