FUNCTIONS

TOPICS:

1. DEFINITION

   A function is a block of code that performs a specific task

2. ADVANTAGES
   a. Code reusability
   b. Easy to maintain
   c. Easy to debug
   d. Reduces the code repetition

3. TYPES OF FUNCTIONS

   There are mainly two types:

   a. Built-in functions:

      The functions which comes from python software is built-in functions

   b. User-defined functions

      The functions which are created by the programmer is user-defined functions

4. SYNTAX AND EXPLANATION

   SYNTAX:

   Defining a function:

```
def function_name(parameters):
        statements
        return expression
```

   Calling a function:

```
variable=function_name()
```

   EXPLANATION:

   ⇨ def and function_name are mandatory

   ⇨ parameters, statements and return expression are optional

   ⇨ parameters are the input to the function, while calling the function we have to pass the values to the parameters

   ⇨ no.of parameters and no.of values should be balanced

   ⇨ return should be the last statement in the function

- ⇨ the statements after the return statement will not be executed
- ⇨ return statement will return an expression to the calling function
- ⇨ simply return should be used to exit from the function
- ⇨ in python, function will return multiple values

5. TYPES OF ARGUMENTS
   a. Positional arguments

```
def some(name,age):
        print(name,age)
some("raju",30)
```

   b. Keyword arguments

```
def some(name,age):
        print(name,age)
some(name="raju",age=30)
```

   - ⇨ we can mix the both positional and keyword arguments but keyword argument should be the last

   c. Default arguments

```
def some(name,age=30):
        print(name,age)

some(name="raju",age=40)

some(name="ram")
```

   - ⇨ default parameter should be the last

d. Variable length arguments

- ⇨ It is represented by *variable_name
- ⇨ We can mix both positional and variable length arguments but variable length argument should be the last
- ⇨ If your taking any argument after the variable length argument that should be the keyword argument

Ex:

```
def add(a,*n):
    sum=0
    for i in n:
        sum+=i
    return   sum+a
result=add(10,20,30,40)
print(result)
```

6. LAMBDA FUNCTION

- ⇨ The functions which doesn't have any name is called anonymous/lambda functions
- ⇨ These are used for instant use

SYNTAX:

```
Variablename=lambda argument_list:expression
```

Ex:

```
result=lambda a,b:a+b
print(result(2,3))
```

7. RECURSIVE FUNCTION

⇨ A function that calls itself is known as recursion function

⇨ It reduces the length of code

Ex:

```
num=int(input("enter the number"))
def fact(num):
    if num==0:
        return 1
    elif num<0:
        return -1
    else:
        return num*fact(num-1)
fact_result=fact(num)
print(f"factorial of {num}={fact_result}")
```

8. TYPES OF VARIABLES

a. Local variable

⇨ The variable which is declared with in the function/method/block is known as local variable

⇨ Scope: within the function

⇨ Once function execution is completed the local variable will be removed from the memory

Ex:

```
def localvar():
    i=10
    print(i)
local()
print(i)
```

b. Global variable:

⇨ A variable which is declared outside the function/block is known as global variable

⇨ Scope: within the program

Ex:

```
i=10
def globalvar():
   print(i)
globalvar()
print(i)
```

c. If you want to change the global variable value inside the function we have to use the global keyword

Ex:

```
j=10
def globalvar1():
   global j
   j=11
   print(j)
globalvar1()
```

d. If global and local variable name is same:

⇨ The first priority goes to the local variable

⇨ If you want to access the global variable inside the function we have to use globals() function

```
Syntax: globals()['variable_name']
```

Ex:

```
g=10
def localvar():
    g=11
    print(g)
    print(globals()['g'])
localvar()
```

9. PASS BY VALUE
   a. In this we are passing copy of the variable value to the function
   b. When we try to change the value of the variable then the new object will be created in the memory with the same name, Because in python integers, floats, strings, tuple are immutable

Ex:

```
x=10
def modify(x):
    x=15
    print(id(x))
    print(x)
modify(x)
print(id(x))
print(x)
```

10. PASS BY REFERENCE

a. In this we are passing the reference/address of the variable to the function
b. When we try to modify the data inside the list the same object will be modified and no new object will be created because lists are mutable

Ex:

```
li=[10,20]
def modify(li):
   li.append(30)
   print(id(li))
   print(li)
modify(li)
print(id(li))
print(li)
```