FUNCTIONS

definition:

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

functions allow you to encapsulate logic, making your code modular and easier to manage.

definition & Calling:

```
def greet():
    print("Hello World")

greet()
print('outside function')
```

Output:

```
Hello World
Outside function
```

1. When the function greet() is called, the program control transfer to the function definition.
2. All the code inside the function is executed.
3. The control of the program jumps to

the next statement after the function call.

eg:
```
# function with two arguments                    (Arguments)
def add_numbers(num1, num2):
    sum = num1 + num2
    print('Sum', sum)
# function call with two numbers values
add_numbers(5, 4)
```

Output

Sum : 9

eg.
```
# function definition
def find_square(num):
    result = num * num
    return result

# function call
square = find_square(3)
print('Square', square)
```

The function accept a number and returns the square of the number. The return statement also denotes that the function has ended. Any code after return is not executed.

```
def future-function():
    pass

# this will execute without any action
future-function()
```

PYTHON FUNCTION ARGUMENTS

An argument is a value that is accepted by a function.

```
def add-numbers(a, b):        → Parameters
    sum = a+b
    print('Sum:', sum)

add-numbers(2,3)              → arguments

# output: Sum: 5
```

Here, add_numbers(2,3) specifies that parameters a & b will get values 2 and 3 respectively.

FUNCTION ARGUMENTS WITH DEFAULT VALUES

```
def add-numbers(a=7, b=8):
    sum = a + b
    print('Sum:', sum)
```

# function call with two arguments
add_numbers(2,3)
# function call with one argument
add_numbers(a=2)

# function call with no argument
add_numbers()

Output

Sum: 5
Sum: 10
Sum: 15

PYTHON KEYWORD ARGUMENTS:

In keyword arguments, arguments are assigned based on the name of the arguments.

eg.

```
def display_info(first_name, last_name):
    print('First Name:', first_name)
    print('Last Name:', last_name)

display_info(last_name = 'Ravi',
             first_name = 'Eric')
```

Output:

First Name: Eric
Last Name: Ravi

Position of Arguments doesn't matter

Python function with Arbitrary Arguments:

Sometimes, we do not know in advance the number of arguments that will be passed into a function. To handle this kind of situation, we can use ARBITRARY arguments in Python.

Arbitrary arguments allow us to pass a varying number of values during a function call. We use an asterisk (*) before the parameter name to denote this kind of argument.

~~* arguments~~ * args

eg:

```
# program to find sum of multiple numbers.

def find_sum(*numbers):
    result = 0
    for num in numbers:
        result = result + num
    print("Sum:", result)

# function call with 3 arguments
find_sum(1, 2, 3)

# function call with 2 arguments
```

find_sum (4, 9)

Output :

Sum = 6
Sum = 13

- *args → these arguments are collected into a tuple, which you can then access inside a function.

**kwargs : Python passes variable length non keyword argument to function using *args but we cannot use this to pass keyword argument. For this problem, Python has got a solution called **kwargs, it allows us to pass the variable length of keyword arguments to the function.

In the function we use the double asterisk. ** before the parameter name to denote this type of arguments. The arguments are packed as a dictionary and these argument make a dictionary inside function with name as the parameter excluding double asterisk **.

Output: Data type of argument : <class 'dict'>
Name : R
Age : 30
Data type of argument : <class 'dict'>
Name : L
Age : 27

```
def intro(**data):
    print("\nData type of argument:", type(data))
    for key, value in data.items():
        print("{} is {}".format(key, value))

def person_details(**details):
    print("\nData type of argument:", type(details))
    for key, value in details.items():
        print(f"{key} : {value}")

person_details(Name='R', Age=30)
person_details(Name='L', Age=27, Profession="Doc")
```

PYTHON VARIABLE SCOPE

In Python, we can declare variables in three different scopes:

- Local scope
- global scope
- Non-local scope.

A variable scope specifies the region where we can access a variable.

Python Local Variables

When we declare variables inside a function, these variables will have a local scope (within the function). We cannot access them outside the function.

eg:

```
def greet():
    # local variable
    message = 'Hello'
    print('Local', message)

greet()
# try to access message variable
# outside greet() function
print(message)
```

Output

```
Local Hello
NameError: name 'message' is not defined
```

Here, the message variable is local to the greet() function, so it can only be accessed within the function. That's why we get an error when we try to access it outside the greet() function.

To fix this, we can make the variable named message global.

Python Global Variables:

In Python, a variable declared outside of the function or in global scope, is known as a global variable.

This means that a global variable can be accessed inside or outside of the function.

```
# declare global variable
message = 'Hello'

def greet():
    # declare local variable
    print('Local', message)

greet()
print('Global', message)
```

output:

```
Local Hello
Global Hello
```

Python Recursion :

Recursion is the process of defining something in terms of itself.

A function can call other function. It is even possible for the function to call itself. These type of construct are termed as Recursive functions.

```
def recurse():
    recurse()

recurse()
```

Eg. of a recursive function :

```
def factorial(x):
    if x==1:
        return 1
    else:
        return (x * factorial(x-1))

num = 3
print("The factorial of", num, "is",
      factorial(num))
```

Output

The factorial of 3 is 6

Here, recursion ends, when the number reduces to 1. This is called BASE Condition

Every recursive function must have a base condition that stops the recursion or else the function call itself infinitely.

Python Modules:

Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes, etc.

```
def add(a, b):
    result = a+b
    return result
```

Here, we have defined a function add() inside a module named example.py.

We use import keyword to do this.

```
import example
```

This does not import the names of the functions defined in example directly in the current symbol table. It only imports the module name example there.

Using the module name we can access the function using the dot. operator.

```
example.add(4, 5) # returns 9
```

classmate
Date
Page

IMPORT PYTHON STANDARD LIBRARY MODULES

The Python standard library contains well over 200 modules.

1. math module

- We can use the dir() function to list all the functions names in a module.

eg: print(dir(example))

2. random
3. datetime
4. Pointer
5. OS module
6. Sys module
7. calendar
8.

classmate
Date
Page

BUILT IN FUNCTIONS :

1. abs()
   any()
   all()
   ascii()
   bin()
   bool()
   bytearray()
   input()
   len()
   max()
   min()
   sum()
   sorted()
   .
   .
   .
   etc.

Assignment :

1. Write in built modules available in Python.
2. Write in built functions available in Python
3. Write functions generating Random nos.

1. Write a Python function that takes a list as an argument and returns the sum of all the elements in the list. Test this function with a sample list.

2. Write a Python function named operate that takes two numbers & an operation as arguments. The operation can be add, subtract, multiply or divide. Use lambda functions to perform the respective operations.

1.
```
def sum_of_list(numbers):
    return sum(numbers)

sample_list = [1, 2, 3, 4, 5]
result = sum_of_list(sample_list)
print("The sum of the sample list is :", result)
```