```
# -*- coding: utf-8 -*-
"""
Python Functions
Summary By: Varpe K.M.
Reference: https://www.geeksforgeeks.org/functions-in-python/
           https://www.tutorialspoint.com/python/python_functions.htm

Functions in Python

A function is a set of statements that take inputs,
do some specific computation and produces output.

The idea is to put some commonly or repeatedly
done task together and make a function,
so that instead of writing the same code again and again
for different inputs,
we can call the function.

Python provides built-in functions like print(),etc.
but we can also create your own functions.

These functions are called user-defined functions.
"""

"""
Defining a Function

You can define functions to provide the required functionality.
Here are simple rules to define a function in Python.

    Function blocks begin with the keyword def
    followed by the function name and parentheses ( ( ) ).

    Any input parameters or arguments should be placed within
    these parentheses.
    You can also define parameters inside these parentheses.

    The code block within every function starts with a colon (:)
    and is indented.

    The statement return [expression] exits a function,
    optionally passing back an expression to the caller.
    A return statement with no arguments is the same as return None.

Syntax

def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]

By default, parameters have a positional behavior and
you need to inform them in the same order that they were defined.
Example

The following function takes a string as input parameter and
prints it on standard screen.
"""
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
```

```
   print(str)
   return
"""
Calling a Function

Defining a function only gives it a name,
specifies the parameters that are to be included in the function
and structures the blocks of code.
"""
# Now you can call printme function
#Way 1
s="Hello Function!"
printme(s)

#Way 2
printme("I'm first call to user defined function!")
printme("Again second call to the same function")

# A simple Python function to check
# whether x is even or odd
def evenOdd( x ):
    if (x % 2 == 0):
        print("even")
    else:
        print("odd")

# Driver code
evenOdd(2)
evenOdd(3)

"""
Pass by Reference or pass by value?

One important thing to note is,
in Python every variable name is a reference.
When we pass a variable to a function,
a new reference to the object is created.
"""

# Here x is a new reference to same list lst
def myFun(x):
   x[0] = 20

# Driver Code (Note that lst is modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
myFun(lst);
print(lst)

"""
When we pass a reference and change the received reference
to something else,
the connection between passed and received parameter is broken.
For example, consider below program.
"""
def myFun(x):

   # After below line link of x with previous
   # object gets broken. A new object is assigned
   # to x.
   x = [20, 30, 40]
```

```
    print("Inside myFun =",x)

# Driver Code (Note that lst is not modified
# after function call.
lst = [10, 11, 12, 13, 14, 15]
print("Original Copy =",lst)
myFun(lst);
print("Outside myFun =",lst)

"""
Another example to demonstrate that reference link is broken
if we assign a new value (inside the function).
"""

def myFun(x):

    # After below line link of x with previous
    # object gets broken. A new object is assigned
    # to x.
    x = 20
    print("Inside myFun =",x)

# Driver Code (Note that lst is not modified
# after function call.
x = 10
print("Original Copy =",x)
myFun(x);
print("Inside myFun =",x)

"""
Exercise: Try to guess the output of following code.
"""
def swap(x, y):
    temp = x;
    x = y;
    y = temp;
    print("Inside Swap Function")
    print("x",x)
    print("y",y)

# Driver code
x = 2
y = 3
print("Original Copies x= ",x," y= ",y)

swap(x, y)
print("Outside Swap Function")
print("x",x)
print("y",y)

"""
Default arguments:
A default argument is a parameter that assumes a default value
if a value is not provided in the function call for that argument.
The following example illustrates Default arguments.
"""

# Python program to demonstrate
# default arguments
def myFun(x, y=50):
    print("x: ", x)
```

```
    print("y: ", y)

# Driver code (We call myFun() with only
# argument)
myFun(10)

"""
Like C++ default arguments,
any number of arguments in a function
can have a default value.
But once we have a default argument,
all the arguments to its right must also have default values.
"""
def myFun(x, y=50):
    print("x: ", x)
    print("y: ", y)

myFun(10,20)

def myFun(x, y,z):
    print("x: ", x)
    print("y: ", y)
    print("z: ", z)

myFun(10,20,30)

def myFun(x, y,z=100):
    print("x: ", x)
    print("y: ", y)
    print("z: ", z)

myFun(10,20)

"""
Keyword arguments:
The idea is to allow caller to specify argument name
with values so that caller does not need to remember
order of parameters.
"""
# Python program to demonstrate Keyword Arguments
def student(firstname, lastname):
     print(firstname, lastname)


# Keyword arguments
student(firstname ='Python', lastname ='Programmar')
student(lastname ='VIT', firstname ='Scholar')

"""
Variable length arguments:
We can have both normal and keyword variable number of arguments.
Please see this for details.
"""

# Python program to illustrate
# *args for variable number of arguments
def myFun(*argv):
    for arg in argv:
        print (arg)

myFun('Hello', 'Welcome', 'to', 'VITPune')
```

```
# Python program to illustrate
# *kargs for variable number of keyword arguments

def myFun(**kwargs):
    for key, value in kwargs.items():
        print ("%s == %s" %(key, value))

# Driver code
myFun(first ='Geeks', mid ='for', last='Geeks')

"""
Anonymous functions:
In Python, anonymous function means that a function is without a name.
As we already know that def keyword is used
to define the normal functions and
the lambda keyword is used to create anonymous functions.
Please see this for details.
"""

# Python code to illustrate cube of a number
# using labmda function

cube = lambda x: x*x*x
print(cube(7))

# Function definition is here
sum = lambda arg1, arg2: arg1 + arg2;

# Now you can call sum as a function
print("Value of total : ", sum( 10, 20 ))
print("Value of total : ", sum( 20, 20 ))

"""
The return Statement

The statement return [expression] exits a function,
optionally passing back an expression to the caller.
A return statement with no arguments is the same as return None.

All the above examples are not returning any value.
You can return a value from a function as follows –
"""
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2
   print("Inside the function : ", total)
   return total;

# Now you can call sum function
total = sum( 10, 20 );
print("Outside the function : ", total)

"""
Scope of Variables

All variables in a program may not be accessible
at all locations in that program.
This depends on where you have declared a variable.
```

The scope of a variable determines the portion of
the program where you can access a particular identifier.
There are two basic scopes of variables in Python –

    Global variables
    Local variables

Global vs. Local variables

Local variables
defined inside a function body have a local scope,
This means that local variables can be accessed only inside the function
in which they are declared

Global variables
defined outside of a function body have a global scope.
global variables can be accessed throughout the program
body by all functions.

When you call a function,
the variables declared inside it are brought into scope.
Following is a simple example –

```
"""
total = 0; # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print("Inside the function local total : ", total)
   return total;

# Now you can call sum function
sum( 10, 20 );
print("Outside the function global total : ", total)
print("Sum returned value = ",sum(50,40))
```