Functions in Python

Introduction.

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Objectives

Objectives by the end of this topic you should be able to:

- Calling Functions
- Functions-related Statements Or Expressions
- Advantages Of Python Functions

Learning activities

Learning Activity 10.1: Reading

Read further on user defined functions

Learning Activity 10.2: Journal

Write a Python function to calculate the factorial of a number (a non-negative integer). The function accepts the number as an argument.

Learning Activity 10.3: Discussion

Write a Python function to multiply all the numbers in a list

Assessment

Topic resources

1. The Python Tutorial¶. (n.d.). Retrieved from https://docs.python.org/3/tutorial/index.html
2. Mueller, J. P. (n.d.). *Beginning Programming with Python For Dummies*. S.l.: For Dummies.
3. (n.d.). Python 3.7.4 documentation. Retrieved from https://docs.python.org/3
4. (n.d.). Git Handbook. Retrieved from https://guides.github.com/introduction/git-handbook/
5. Shaw, Z. (2017). *Learn Python 3 the hard way: a very simple introduction to the terrifyingly beautiful world of computers and code*. Boston: Addison-Wesley.
6. Bader, D. (2018). *Python tricks: the book*. Vancouver, BC: Dan Bader.
7. Downey, A. B. (2015). *Think Python*. Sebastopol: OReilly.

8. Ramalho, L. (2016). *Fluent Python:*Beijing: OReilly.

URL Links

https://www.w3schools.in/python-tutorial/functions/
https://data-flair.training/blogs/python-function/
https://www.tutorialspoint.com/python/python_functions.htm

Python Functions

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.

```
# 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)
```

Output:

```
even

odd
```

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. Parameter passing in Python is same as reference passing in Java.

```
# 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)
```

Output:

```
[20, 11, 12, 13, 14, 15]
```

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]

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

Output:

```
[10, 11, 12, 13, 14, 15]
```

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

# Driver Code (Note that lst is not modified
# after function call.
x = 10
myFun(x);
print(x)
```

Output:

```
10
```

Exercise: Try to guess the output of following code.

```
def swap(x, y):
    temp = x;
    x = y;
    y = temp;

# Driver code
x = 2
y = 3
swap(x, y)
print(x)
print(y)
```

Output:

```
2

3
```

Default arguments:

Python allows function arguments to have default values. If the function is called without the argument, the argument gets its default value.

Default Arguments:

Python has a different way of representing syntax and default values for function arguments. Default values indicate that the function argument will take that value if no argument value is passed during function call. The default value is assigned by using assignment(=) operator of the form `keywordname=value`.

Let's understand this through a function *student*. The function *student* contains 3-arguments out of which 2 arguments are assigned with default values. So, function *student* accept one required argument (firstname) and rest two arguments are optional.

```
def student(firstname, lastname ='Mark', standard ='Fifth'):

    print(firstname, lastname, 'studies in', standard, 'Standard')
```

We need to keep the following points in mind while calling functions:

1. In case of passing keyword argument, order of arguments is not important.
2. There should be only one value for one parameter.
3. The passed keyword name should match with the actual keyword name.
4. In case of calling function containing non-keyword arguments, order is important.

Example #1: **Calling functions without keyword arguments**

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')

# 1 positional argument
student('John')

# 3 positional arguments
student('John', 'Gates', 'Seventh')

# 2 positional arguments
student('John', 'Gates')
student('John', 'Seventh')
```

Output:

```
John Mark studies in Fifth Standard

John Gates studies in Seventh Standard

John Gates studies in Fifth Standard

John Seventh studies in Fifth Standard
```

In the first call, there is only one required argument and the rest arguments use the default values. In the second call, lastname and standard arguments value is replaced from default value to new passing value. We can see order of arguments is important from 2nd, 3rd, and 4th call of function.

Example #2: **Calling functions with keyword arguments**

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')

# 1 keyword argument
student(firstname ='John')

# 2 keyword arguments
student(firstname ='John', standard ='Seventh')

# 2 keyword arguments
student(lastname ='Gates', firstname ='John')
```

Output:

```
John Mark studies in Fifth Standard

John Mark studies in Seventh Standard

John Gates studies in Fifth Standard
```

In the first call, there is only one required keyword argument. In the second call, one is required argument and one is optional(standard), whose value get replaced from default to new passing value. In the third call, we can see that order in keyword argument is not important.

Example #3: **Some Invalid function calls**

```
def student(firstname, lastname ='Mark', standard ='Fifth'):
    print(firstname, lastname, 'studies in', standard, 'Standard')

# required argument missing
student()

# non keyword argument after a keyword argument
student(firstname ='John', 'Seventh')

# unknown keyword argument
student(subject ='Maths')
```

Above code will throw error because –

- In the first call, value is not passed for parameter *firstname* which is the required parameter.
- In the second call, there is non-keyword argument after a keyword argument.
- In the third call, the passing keyword argument is not matched with the actual keyword name arguments.

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 ='Geeks', lastname ='Practice')
student(lastname ='Practice', firstname ='Geeks')
```

Output:

```
('Geeks', 'Practice')

('Geeks', 'Practice')
```

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', 'GeeksforGeeks')
```

Output:

```
Hello

Welcome

to

GeeksforGeeks
```

```python
# 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')
```

Output:

```
last == Geeks

mid == for

first == 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.

It has the following syntax:

Syntax

```
lambda arguments : expression
```

- This function can have any number of arguments but only one expression, which is evaluated and returned.
- One is free to use lambda functions wherever function objects are required.
- You need to keep in your knowledge that lambda functions are syntactically restricted to a single expression.
- It has various uses in particular fields of programming besides other types of expressions in functions.

Example #1:

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

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

Output:

```
343
```

Example #2:

```
# Python program to demonstrate
# lambda functions


x ="GeeksforGeeks"

# lambda gets pass to print
(lambda x : print(x))(x)
```

Output

```
GeeksforGeeks
```

Example #3: Difference between lambda and normal function call

```
# Python program to illustrate cube of a number
# showing difference between def() and lambda().


def cube(y):
    return y*y*y;

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

print(cube(5))
```

Output

```
343

125
```

Example #4: The lambda function gets more helpful when used inside a function.

```python
# Python program to demonstrate
# lmabda functions


def power(n):
    return lambda a : a ** n

# base = lambda a : a**2 get
# returned to base
base = power(2)

print("Now power is set to 2")

# when calling base it gets
# executed with already set with 2
print("8 powerof 2 = ", base(8))

# base = lambda a : a**5 get
# returned to base
base = power(5)
print("Now power is set to 5")

# when calling base it gets executed
# with already set with newly 2
print("8 powerof 5 = ", base(8))
```

Output

```
Now power is set to 2

8 powerof 2 =  64

Now power is set to 5

8 powerof 5 =  32768
```

We can also replace list comprehension with Lamba by using a map() method, not only it is a fast but efficient too and let's also see how to use lambda in the filter().

Example #5: filter() and map()

```python
# Python program to demonstrate
# lambda functions inside map()
# and filter()


a = [100, 2, 8, 60, 5, 4, 3, 31, 10, 11]

# in filter either we use assignment or
# conditional operator, the pass actual
# parameter will get return
filtered = filter (lambda x: x % 2 == 0, a)
print(list(filtered))

# in map either we use assignment or
# conditional operator, the result of
# the value will get returned
maped = map (lambda x: x % 2 == 0, a)
print(list(maped))
```

Output

```
[100, 2, 8, 60, 4, 10]

[True, True, True, True, False, True, False, False, True, False]
```

Revision questions

1. Write a Python function to shuffle and print a specified list. Go to the editor
Click me to see the sample solution (convert it to a function)

2. Write a Python function to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included). Go to the editor
Click me to see the sample solution (convert it to a function)

3. Write a Python function to generate and print a list except for the first 5 elements, where the values are square of numbers between 1 and 30 (both included). Go to the editor
Click me to see the sample solution (convert it to a function)

4. Write a Python function to generate all permutations of a list in Python. Go to the editor
Click me to see the sample solution (convert it to a function)

5. Write a Python function to get the difference between the two lists. Go to the editor
Click me to see the sample solution(convert it to a function)

6. Write a Python function to access the index of a list. Go to the editor
Click me to see the sample solution (convert it to a function)

7. Write a Python function to convert a list of characters into a string. Go to the editor
Click me to see the sample solution (convert it to a function)