Ch03-3-Functions-UserDefined

September 10, 2025

1 User-defined Functions

- http://openbookproject.net/thinkcs/python/english3e/functions.html

1.1 Topics

- how to define and use your own functions
- variables scopes
- how to pass data to functions (by value and reference)
- how to return data from functions
- unit or functional testing with assert

1.2 Functions

- named sequence of statements that execute together to solve some task
- primary purpose is to help us break the problem into smaller sub-problems or tasks
- two types: fruitful and void/fruitless functions
- must be defined before it can be used or called (two step process)
- concept of function is burrowed from Algebra
- e.g.

  Let's say: $f(x) = x^2 + x + 1$

  $y = f(1) = 1 + 1 + 1 = 3$

  $y = f(-2) = 4 - 2 + 1 = 3$

1.2.1 Two-step process

1. Define a function
2. Call or use function

1.2.2 syntax to define function

```
def functionName( PARAMETER1, PARAMETER2, ... ):
    # STATEMENTS
    return VALUE
```

- PARAMETERS and return statements are OPTIONAL

- function NAME follows the same rules as a variable/identifier name
- recall some built-in functions and object methods have been used in previous chapters...

1.2.3 syntax to call function

- call function by its name
- use return value(s) if any

```
VARIABLE = functionName( ARGUMENT1, ARGUMENT2, ...)
```

1.3 Why functions?

dividing a program into functions or sub-programs have several advantages: - give you an opportunity to name a group of statements, which makes your program easier to read and debug - can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place - allow you to debug the parts one at a time (in a team) and then assemble them into a working whole - write once, test, share, and reuse many times (libraries, e.g.)

1.4 Types of functions

- two types: fruitful and fruitless functions

1.5 Fruitless functions

- also called void functions
- they do not **explictly** return a value

[1]:
```
# Function definition
# function prints the result but doesn't explictly return anything
def greet():
    print('Hello World!')
```

[2]:
```
# Function call
greet()
greet()
```

```
Hello World!
Hello World!
```

[3]:
```
# void/fruitless function; returns None by default
a = greet() # returned value by greet() assigned to a
print('a =', a)
```

```
Hello World!
a = None
```

[4]:
```
type(greet)
```

[4]:
```
function
```

[5]:
```
# function can be assigned to another identifier
myfunc = greet
type(myfunc)
```

[5]:
```
function
```

[6]:
```
myfunc()
```

```
Hello World!
```

1.6 Fruitful functions

- functions that explictly return some value(s) using **return** statement
- more useful functions
- answer returned can be used as intermediate values to solve bigger problems
- can be used and tested independently
- fruitful functions usually take some arguments and return value(s) as answer
- most built-in and library functions are fruitful
- typically return is the last statement to execute; but not necessarily
- function returns back to the caller immidiately after return statement is executed
  - will skip code if any exists after return statement

[10]:
```
# fruitful function
def getName():
    name = input("Hi there, enter your full name: ")
    return name
    print(f'Hi {name}, nice meeting you!') # dead code - will not be executed
```

[11]:
```
userName = getName()
```

```
Hi there, enter your full name: John Smith
```

[13]:
```
userName
```

[13]:
```
'John Smith'
```

[15]:
```
print(f'Hi {userName}, nice meeting you!')
```

```
Hi John Smith, nice meeting you!
```

1.7 Passing data as arguments to functions

- functions are subprograms that may need external data to work with
- you can pass data to functions via parameters/arguments
- can provide 1 or more parameters to pass 1 or more data
- can provide default values to parameters
  - makes the parameter optional when the function is called
- if a function has a required parameter, data must be provided for each required parameter!
  - otherwise, you'll get error!

1.7.1 Visualize with PythonTutor.com

[16]:
```python
# Function takes one required argument
def greet(name):
    print(f'Hello {name}')
```

[17]:
```python
# Pass 'John Smith' literal value as an argument for name parameter
greet('John Smith')
```

```
Hello John Smith
```

[18]:
```python
greet('Jane')
```

```
Hello Jane
```

[19]:
```python
# Arguments can be variables as well
n = 'Michael Smith'
greet(n)
```

```
Hello Michael Smith
```

[21]:
```python
n1 = input('Enter your name: ')
greet(n1)
```

```
Enter your name: Jake Jones
Hello Jake Jones
```

[22]:
```python
greet()
# How to fix? provide either default value or call it properly
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[22], line 1
----> 1 greet()

TypeError: greet() missing 1 required positional argument: 'name'
```

[23]:
```python
# function takes one optional argument
def greet(name="Anonymous"):
    print(f'Hello, {name}')
```

[24]:
```python
# calling greet without an argument
# default value for name will be used!
greet()
```

```
Hello, Anonymous
```

[25]:
```python
greet('adfasd')
```

```
Hello, adfasd
```

[26]:
```python
user = input('Enter your name: ')
greet(user) # calling greet with an argument
```

```
Enter your name: Jackie Chain
Hello, Jackie Chain
```

1.7.2 Visualize in PythonTutor.com

1.7.3 Exercise

- Define a function that takes two numbers as arguments and returns the sum of the two numbers as answer

[29]:
```python
def add(num1, num2):
    """ add function

    Take two numeric values: num1 and num2.
    Calculate and return the sum of num1 and num2.
    """
    total = num1 + num2
    return total
```

[30]:
```python
# displays the function prototype and docstring below it
help(add)
```

```
Help on function add in module __main__:

add(num1, num2)
    add function

    Take two numeric values: num1 and num2.
    Calculate and return the sum of num1 and num2.
```

[31]:
```python
import math
help(math.sin)
```

```
Help on built-in function sin in module math:

sin(x, /)
    Return the sine of x (measured in radians).
```

[32]:
```python
# Test add function
print(add(100, 200))
```

```
300
```

[33]:
```python
t = add(100.99, -10)
print('sum = ', t)
```

```
sum =  90.99
```

[34]:
```python
num1 = 15
num2 = 10.5
total = add(num1, num2)
print(f'{num1} a+ {num2} = {total}')
```

```
15 a+ 10.5 = 25.5
```

1.7.4 Exercise

- Define a function that takes two numbers and returns the product of the two numbers.

[35]:
```python
# Exercise - complete the following function
def multiply(x, y):
    """
    Function take two numbers: x and y.
    Return the product of x and y.
    """

    # FIXME
    pass
```

[36]:
```python
# Help can be run for user-defined functions as well
help(multiply)
```

```
Help on function multiply in module __main__:

multiply(x, y)
    Function take two numbers: x and y.
    Return the product of x and y.

```

[37]:
```python
# Manually test multiply function
```

1.8 Ways of passing data to functions

- data/values are passed to functions in two ways

1.8.1 pass by value

- fundamental types and literals (string, int, float) are passed by value
  - values passed as arguments are copied to the corresponding parameters

1.8.2 pass by reference

- advanced container types (tuple, list, dict, etc.) are passed by reference

    – parameters and corresponding arguments become alias pointing to the same memory location
- this topic will be discussed in the corresponding chapter covering those container types

[44]:
```
# Pass by value demo
var1 = 'John' # Global variable

def greetSomeone(para1):
    print('hello', para1)
    var1 = 'Jake' # Local variable
    print('hello again', para1)

greetSomeone(var1)
print('var1 = ', var1)
```

```
hello John
hello again John
var1 =  John
```

1.8.3 visualize pass by value with PythonTutor.com

1.9 Fruitful functions returning multiple values

- functions can return more than 1 values
- multiple comma separated values can be returned
- the values are return as Tuple type (more on this later)

[45]:
```
def findAreaAndPerimeter(length, width):
    """
    Take length and width of a rectangle.
    Find and return area and perimeter of the rectangle.
    """

    area = length*width
    perimeter = 2*(length+width)
    return area, perimeter
```

[46]:
```
print(findAreaAndPerimeter(10, 5))
```

```
(50, 30)
```

[47]:
```
a, p = findAreaAndPerimeter(20, 10)
print(f'area = {a} and perimeter = {p}')
```

```
area = 200 and perimeter = 60
```

1.10 Function calling a function

- a function can be called from within another function
- a function can call itself – called recursion (see Chapter 13)

[49]:
```python
def average(num1, num2):
    sum_of_nums = add(num1, num2)
    return sum_of_nums/2
```

[50]:
```python
avg = average(10, 20)
print(f'avg of 10 and 20 = {avg}')
```

```
avg of 10 and 20 = 15.0
```

1.11 Exercises

1.11.1 exercise 1

Write a function that takes two numbers; subtracts the second from the first and returns the difference. Write two test cases.

[51]:
```python
# Solution to exercise 1
def sub(num1, num2):
    return num1 - num2
```

1.11.2 exercise 2

Write a function that converts seconds to hours, minutes and seconds. Function then returns the values in **HH:MM:SS** format (e.g., 01:09:10)

[54]:
```python
def get_time(seconds):
    pass
```

1.11.3 exercise 3

Write a function called hypotenuse that returns the length of the hypotenuse of a right triangle given the lengths of the two legs as parameters.

[57]:
```python
def hypotenuse(leg1, leg2):
    pass
```

1.11.4 exercise 4

Write a function $slope(x1, y1, x2, y2)$ that returns the slope of the line through the points $(x1, y1)$ and $(x2, y2)$.

Then use a call to slope in a new function named intercept(x1, y1, x2, y2) that returns the y-intercept of the line through the points $(x1, y1)$ and $(x2, y2)$

[60]:
```python
def slope(x1, y1, x2, y2):
    pass
```

[63]:
```python
def intercept(x1, y1, x2, y2):
    pass
```

1.12 Kattis problems requiring functions

- functions are not required to solve problems
- you can use function to solve each and every problem or not use one
- function is required if you must write automated unit tests
- function is recommended for breaking a problem into smaller sub-problems and making the solution modular

[ ]:

```

```