Functions

There are many other learning and practice resources on the web.

Examples are [www.w3schools.com](http://www.w3schools.com) [www.freecodecamp.org](http://www.freecodecamp.org) and [www.realpython.com](http://www.realpython.com)

# Section – creating functions

## Part 1

Create and save these functions in a python file named starter.py

def two():

""" Always return the integer 2 """

return 2

def times2(a\_value):

""" Return twice a\_value """

the\_result = a\_value + a\_value

return the\_result

def times3(a\_value):

""" Return thrice a\_value """

return a\_value \* 3

def first\_last(a\_string):

return a\_string[0] + a\_string[-1]

Test them to make sure they work as anticipated.

## Part 2

There are four temperature scales in common use: **K**elvin, **C**entigrade, **R**ankine, **F**ahrenheit.

The conversion factors between these temperature scales are:

|  |  |  |
| --- | --- | --- |
| From | To | Formula |
| Kelvin | Rankine | Kelvin \*9/5 |
| Rankine | Fahrenheit | Rankine - 459.67 |
| Fahrenheit | Celsius | (Fahrenheit -32) \*5/9 |
| Celsius | Kelvin | Centigrade + 273.15 |

Your task is to design, write and test a function for each of these conversions. Call them k2r(), r2f(), f2c() and c2k(). Save them all in the same file called temp\_conv.py

# Section 2 – scope

Open a new python script and call it scope.py and save this program into it:

# define a variable that is external to all functions

a = 5

Now explore what happens when we use the same name INSIDE # function definitions

def scope1():  
 return a

Function scope1() returns the value of the variable 'a' that exists only in its external namespace"""

def scope2():  
 a = 27  
 return a

Function scope2() defines and return an internal variable 'a'. This masks/hides the external name 'a' and its value. After executing this function the value of external 'a' is unaffected"""

def scope3(): # throws syntax error when called  
 a = a + 1  
 return a

Function scope3() attempts to increment value of a variable named 'a' but this fails with syntax error "local variable 'a' referenced before assignment".

Reading the expression left to right (as Python does) we are asking Python to create a variable named 'a' which masks external 'a' and then assign it a value of one more than its current value. But at that point the new 'a' doesn't properly exist and certainly doesn't have a value - hence Python objects that we are trying to use (reference) 'a' before it has been assigned a value.

def scope4(): # using the keyword ‘global’  
 global a  
 a = a + 1  
 return a

The ‘global’ keyword is a rich source of bugs so please AVOID using it.

Use of the 'global' keyword links the internal name 'a' to the external scope. The variable name 'a' can then be used and assigned values but it is changing the external 'a' at the same time as the local value. Check the value of external 'a' after running this function.

Execute scope.py and use dir() to see what is in memory. Then execute these functions in IDLE and explain the results you get:

>>> scope1()

>>> scope2()

>>> scope3()

>>> scope4()

Now delete the variable a from memory in IDLE

>>> del(a)

>>> scope1()

Explain why it did execute but now it doesn’t

# Section 3– reusing functions

## Part 1

Open the python file starter.py in IDLE and add these functions:

def times5(a\_number):

""" Return five times a\_number """

return times2(a\_number) + times3(a\_number)

def times6(a\_number):

""" Return six times a\_number """

return times2( times3( a\_number ) )

Test that they work correctly.

## Part 2

Your task is to design and write the functions k2c() and f2r() which convert temperatures from Kelvin to Celsius and from Fahrenheit to Rankine by reusing ONLY the temperature conversion functions you built in part 1.

Open the python file temp\_conv.py in IDLE, add these functions, and test that they work.

# Section 4– parameters

## Part 1

Create and save these functions in a python file named params.py

def parms\_v1(num1, num2, num3):

""" there can be any number of parameters """

return num1 + (num2 \* num3)

def parms\_v2(num1, num2, num3 = 1):

""" Third parameter is optional """

return num1 + (num2 \* num3)

Test them to make sure they work as anticipated. For instance:

parms\_v2(4,5,6),

parms\_v2(4,5),

parms\_v2(1, num3 = 0, num2 = 11),

parms\_v2(0, 17, 2),

parms\_v2(num2 = 17, num3 = 2, num1 = 0)

Check the values returned to be sure you understand what is happening.

## Part 2 – parameters can be any type

Parameter values can be any type of object, including other functions.

In this example the default value of the parameter hash is the built-in function hash()

def mod00hash(object1, hash=hash):

return hash(str1)%100

Create and save this function in the file params.py new python file.

Then execute these calls to the function to confirm it is working correctly:

mod00hash (5)

mod00hash (’5’)

mod00hash (‘six’)

mod00hash (‘six’, hash)

mod00hash (’six’, id) # the built-in function id()

Explain the results you get.

# Section 5 – return values

Part 1 – none, one or many return values

Create and save these functions in a python file named return\_values.py

def no\_return():

pass

def empty\_return():

return

def single\_return():

return ‘single value returned’

def many\_return():

return ‘as’, ‘many’, ‘values’, ‘as’, ‘is’, ‘needed’

Execute the python script and test that the functions work.

Determine the type of the values returned by these functions using the built-in function type()

type(no\_return())

type(empty\_return())

type(single\_return())

type(many\_return())

Explain the following statements and the values they return:

len(single\_return())

single\_return()[7:12]

len(many\_return())

many\_return()[2][3]

## Part 2 – functions can return any object

This means that functions can create and return other functions. Add this function to return\_values.py:

def nopoints():

def inner\_function():

return 'zero points'

return inner\_function

Now execute these statements and explain the results

f = nopoints()

type(f)

type(f())

f()

# Section 6 – importing and reusing code

## Part 1 – reusing your code

Create a new python file called starter2.py and import the module starter.py you created earlier with the statement:

import starter

After running that module in IDLE then evaluate these two expressions:

dir() # the name ‘starter’ is visible

dir(starter) # lists objects in object with name ‘starter’.

We can now use these imported objects (functions in this case) in this module, starter2.py:

def times4(a\_value):

"""demonstrate use of imported function"""

return starter.times2(a\_value)+starter.times2(a\_value)

times7(a\_value):

pass # you do this

times11(a\_value):

pass # you do this

times11(a\_value):

pass # you do this

Test that they work.

## Part 2 – the python turtle (fun)

Create a new python script and call it turtle1.py and add this fun piece of code:

from turtle import \*

def sq(pencolor, fillcolor):

color(pencolor, fillcolor)

begin\_fill()

forward(200)

left(90)

forward(200)

left(90)

forward(200)

left(90)

forward(200)

end\_fill()

Use IDLE to execute this program and test it works by calling the function sq()

>>> sq('blue', 'green') # choose any standard color

There are many fun Turtle tutorials on the web.

## Part 3 – random

Use the random package to create the function dice() which simulates throwing a many sided dice:

import random # from the standard library

def dice(sides=6):

return random.randint(1,sides)

Create a new python script called dice.py and write and test the dice() function.

dice()

# call dice() several times to confirm it only   
# returns integer values between 1 and 6

dice(10)

# call dice(10) several times to confirm it only   
# returns integer values between 1 and 10

In the IDLE interactive shell execute these statements

>>> import random

>>> fruits = ‘apple pear orange mango melon grape’

Then execute this statement several times

>>> random.choice(fruits.split())

Explain what is going on.

# Section – more practice

Go to [www.w3schools.com](http://www.w3schools.com) and work through the functions tutorial and tests <https://www.w3schools.com/python/python_functions.asp>

By now you should be familiar with most of the content within the following sections of W3Schools:

<https://www.w3schools.com/python/python_comments.asp>

<https://www.w3schools.com/python/python_variables.asp>

<https://www.w3schools.com/python/python_numbers.asp>

<https://www.w3schools.com/python/python_casting.asp>

<https://www.w3schools.com/python/python_strings.asp>

<https://www.w3schools.com/python/python_booleans.asp>

<https://www.w3schools.com/python/python_operators.asp>

<https://www.w3schools.com/python/python_functions.asp>