

16-functions

October 25, 2020

1 Functions

So far in the course you have used functions to code your solutions. Functions are one of the main components of programs, and a big computer program is usually a collection of many functions that interact between each other. A good function is one that:

- serves a specific purpose
- produces an output for every input
- provides a functionality that can be reused in many other parts of the code

For example, in several exercises in this course you needed to find out if a number is prime or not. Instead of copying and pasting that piece of code from one place to the other, the correct thing to do is to create a function, and *call* this function at the places you need.

```
[1]: # Function definition
def isPrime(n):
    if n < 2:
        return False

    for i in range(2,n):
        if n % i == 0:
            return False

    return True

def allPrimesUpTo(n):
    primes = []
    for i in range(n):
        if isPrime(i): # Function call
            primes += [i]

    return primes
```

Many times solving a problem involves splitting it into smaller parts, solving the parts, and combining these into a solution for the bigger problem. It might help to implement a function for each of the smaller parts. The advantages of this approach are:

- Testing each function separately is easier than the entire program (especially if a function is doing a very specific task, with clear input and output)

- This avoids the propagation of errors. If you made a mistake when writing a function, fixing it involves change only that function.
- Your code is more structured, clear, elegant, and easier to read. It is typically smaller.

Finding how to organize your code into a set of functions is a little bit like art. You need to be able to find beauty and function in order.

1.1 Function lexicon

We have used a lot of the words related to functions during the course. Let's give them a precise meaning now.

- **Function definition:** the definition of the function (`def isPrime(n): ...` above)
- **Function call:** when a function is used, or invoked (`isPrime(i)` inside the loop above)
- **Parameters:** variables that are the input in the function definition (`n` in `def isPrime(n)`)
- **Arguments:** input passed to a function when it is called (`i` in `isPrime(i)`)
- **Return value:** the value returned by a function call
- **Return type:** type of the value returned by a function
- **Local variable:** variables declared inside functions
- **Global variable:** variables declared outside functions

1.1.1 More about local and global variables

So far we have only dealt with *local* variables. Since our programs consisted of only one function, there was no need to have *global* variables shared by all functions. If many functions need to access the same information, and this cannot be easily passed as a parameter, it is sometimes useful to keep this information in a global variable.

In this course, you will not be required to use global variables, but feel free to do that if you think this will make your life easier. **Be careful** though. It is very easy to lose track of when and how global variables were modified; and debugging might become a nightmare.

The code below shows how local and global variables work.

```
[4]: def f():
      #x is a local variable, which can be accessed only inside this function.
      x = 10
      print(x)

f()
#Trying to access x from outside the function f() will fail since x is a local_
→variable to f().
print(x)
```

10

NameError

Traceback (most recent call last)

```
<ipython-input-4-e0c96a363ac0> in <module>
    6 f()
    7 #Trying to access x from outside the function f() will fail since x is
↳ a local variable to f().
----> 8 print(x)
```

NameError: name 'x' is not defined

```
[7]: """
x is a global variable, which can be accessed by any
function inside this python file (or module)
"""
x = 10
def f1():
    print(x)

def f2():
    print(x)

f1()
f2()
```

10
10

```
[10]: x = 10
def f():
    #This is a different x that is local to this function (i.e., defined
#within the scope of this function)
    x = 20
    print(x)

f()
print(x)
```

20
10

In the above snippet of code, x is considered as a new local variable in f(). But, what if we want to modify the global variable x in f()? The following code shows how this can be done.

```
[11]: x = 10
def f():
    global x #indicates that the variable x we will use is the global one
    x = 20
    print(x)
```

```
f()
print(x)
```

```
20
20
```

Here is another example:

```
[2]: # A global variable
a = 15

# A function declaration
def f():
    global a # Indicates that the variable a we are using is the global one
    b = a * 10 # Reads a
    a = 100 # Overwrites a (for no good reason)
    return b

print("a before the function call:", a)
print("f() returns:", f())
print("a after the function call:", a)
```

```
a before the function call: 15
f() returns: 150
a after the function call: 100
```

1.2 Interesting facts about functions

1.2.1 Docstring

When defining functions, you can provide them with a help text placed between triple quotes below the first line of the function declaration. This help text, called *docstring* can be accessed by `help(functionName)`. You can try to call `help` on function that you already know to see their docstring.

```
[3]: def f(a, b):
    """Returns the sum of a and b"""
    c = a + b
    return c

help(f)
```

```
Help on function f in module __main__:
```

```
f(a, b)
    Returns the sum of a and b
```

1.2.2 Default parameters

It is possible to define functions with *default* parameters. These are optional parameters that take a default value in case the user does not pass a value in its place.

```
[4]: def increment(x, inc=1):
      return x + inc

      print("No second argument given: ", increment(10))
      print("Second argument given: ", increment(10, 5))
```

```
No second argument given: 11
Second argument given: 15
```

1.2.3 Named parameters

Functions can have *named* parameters, and this way the order of parameters can be changed, or some of them can be omitted (taking their default value). Many functions in the python libraries have named parameters. For example, when you use `sorted(L, reverse=True)`, you are using the named parameter `reverse` from the `sorted` function.

```
[5]: def f(a, b, c=0, d=1):
      return (a + b + c)*d

      print("Using named parameter d and omitting c:", f(1,1,d=5))
```

```
Using named parameter d and omitting c: 10
```

1.2.4 Pass-by-Value vs. Pass-by-Reference

Arguments can be passed to functions either by *value* or by *reference*, **based on the type of the argument**. Pass-by-value implies that only a value of the argument is passed to the function, hence, any change to the value in the function will NOT be reflected back in the passed argument (or variable). In contrary, pass-by-reference entails that a reference to the argument is passed to the function, hence, any change to the value in the function will be reflected back in the passed argument (or variable). Here are some examples on how this works.

```
[13]: def f(x):
      #change x
      x = x + 1

      a = 10
      print(a)
      #integers and floats in Python are passed by value.
      f(a)
      print(a)
```

```
10
10
```

```
[16]: def f(x):
        if len(x) > 0:
            #change x
            x[0] = 20

a = [10]
print(a)
#Lists in Python are passed by reference.
f(a)
print(a)
```

```
[10]
[20]
```

1.3 Example: Sudoku

As an example on how we can use functions to avoid code repetition, let's develop a program that checks if a sudoku solution is valid. Sudoku is a game on a 9x9 board that needs to be completed by filling each position with one of the numbers 1 through 9. Initially the board has already some numbers on it. A solved board is one where each column, each row, and each of the nine 3x3 tiles that compose the board contain all of the digits from 1 to 9.

Since we need to check some conditions for several parts of the board, you can make functions for checking that separate from our main function.

```
[19]: import math

def getColumns(board):
    """This function extracts all the columns in the board and stores them in a
    ↪list of lists, which renders the transpose of the board"""

    columns = []
    for i in range(len(board)):
        c = []
        for j in range(len(board)):
            c.append(board[j][i])

        columns.append(c)

    return columns

def getFile(board, tileNum):
    """This function returns the tile with number tileNum. The 9 tiles in the
    ↪board are numbered 0 to 8"""

    #This is the height (and the width) of the tile, which is 3 (and 3) in our
    ↪case.
```

```

tile_h_w = int(math.sqrt(len(board)))

#This is the row index from where the tile starts on the board.
rowIndex = tileNum // tile_h_w * tile_h_w

#This is the column index from where the tile starts on the board.
colIndex = tileNum - rowIndex * tile_h_w

tile = []

for i in range(rowIndex):
    r = []
    for j in range(colIndex):
        r.append(board[i][j])
    tile.append(r)

return tile

def isRowOrColValid(rc):
    """This function checks the validity of a row or a column based on the
    →sudoku requirements"""
    u = []
    for elem in rc:
        if 1 <= elem <= 9 and elem not in u:
            u.append(elem)
        else:
            return False

    return True

def isTileValid(t):
    """This function checks the validity of a tile based on the sudoku
    →requirements"""
    u = []
    for i in range(len(t)):
        for j in range(len(t[i])):
            if 1 <= t[i][j] <= 9 and t[i][j] not in u:
                u.append(t[i][j])
            else:
                return False
    return True

def isSquare(board):
    """This function verifies whether the board is a 9 * 9 matrix"""

    if len(board) != 9:

```

```

        return False

    for i in range(len(board)):
        if len(board[i]) != 9:
            return False

    return True

def checkSudoku(board):
    if isSquare(board) == False:
        return False

    allCols = getColumns(board)

    for i in range(len(board)):
        if isRowOrColValid(board[i]) == False or isRowOrColValid(allCols[i]) ==
→False:
            return False

    for j in range(len(board)):
        if isTileValid(getTile(board, j)) == False:
            return False

    return True

'''
The board below represents this one:

2 4 5 | 9 8 1 | 3 7 6
1 6 9 | 2 7 3 | 5 8 4
8 3 7 | 5 6 4 | 2 1 9
-----+-----+-----
9 7 6 | 1 2 5 | 4 3 8
5 1 3 | 4 9 8 | 6 2 7
4 8 2 | 7 3 6 | 9 5 1
-----+-----+-----
3 9 1 | 6 5 7 | 8 4 2
7 2 8 | 3 4 9 | 1 6 5
6 5 4 | 8 1 2 | 7 9 3
'''

board = [
    [2,4,5,9,8,1,3,7,6],
    [1,6,9,2,7,3,5,8,4],
    [8,3,7,5,6,4,2,1,9],
    [9,7,6,1,2,5,4,3,8],

```



```
[5,1,3,4,9,8,6,2,7],
[4,8,2,7,3,6,9,5,1],
[3,9,1,6,5,7,8,4,2],
[7,2,8,3,4,9,1,6,5],
[6,5,4,8,1,2,7,9,3] ]
```

```
checkSudoku(board)
```

[19]: True

1.4 Example: image processing

On a previous lab you were required to implement functions that manipulate an image. The skeleton of all functions were more or less the same. The only thing that changed was the kind of transformation you did to the (r,g,b) values of each pixel.

The transformation of each pixel can be implemented using a function, and this function can be passed as an argument to a function just like a normal variable. See an example in the code below.

```
[7]: import matplotlib.pyplot as plt
from matplotlib.pyplot import imread

# Transform function takes a function f as a parameter
def transform(img, f):
    rows = len(img)
    cols = len(img[0])
    new_img = []
    for i in range(rows):
        new_row = []
        for j in range(cols):
            r = img[i][j][0]
            g = img[i][j][1]
            b = img[i][j][2]
            # f is used to transform the previous (r,g,b) value into the new
            →pixel
            new_pixel = f(r, g, b)
            new_row += [new_pixel]
        new_img += [new_row]
    return new_img

def makeTransparent(r,g,b):
    return (r,g,b,75)

def makeGrayscale(r,g,b):
    return 0.3*r + 0.59*g + 0.11*b

def removeRed(r,g,b):
```

```
return (0, g, b)

##### Code for showing the picture #####

# Loads the image
img = imread('doha.png') # Change this to a file in your computer so that it
→works

# The function can be called using the function we want
new_image = transform(img, removeRed)

# Uncomment to test the transform function
plt.imshow(new_image)

plt.show()
```

