Lab 8: Advanced Functions

CSE/IT 107

NMT Computer Science

"All thought is a kind of computation."

— D. Hobbes

"Vague and nebulous is the beginning of all things, but not their end."

— K. Gibran

"It [programming] is the only job I can think of where I get to be both an engineer and artist. There's an incredible, rigorous, technical element to it, which I like because you have to do very precise thinking. On the other hand, it has a wildly creative side where the boundaries of imagination are the only real limitation."

— A. Hertzfeld

---

1 Introduction

2   Advanced Functions

2.1   Default Arguments

Many of the built-in functions we have been using, such as `range` or `input`, accept a variable number of arguments. We can do the same thing with our own funcitions by specifying default values for our arguments:

```python
def celebrate(times=1):
    for i in range(times):
        print("Yay!")

print("First call:")
celebrate()
print("Second call:")
celebrate(5)
```

Note what happens for each call of this function. The first time it only prints `"Yay!"` once, since the default argument is `1`, while the second call overwrites the default value and prints it five times.

While you are allowed to have a function that has some arguments with default values and some without, you must always put those with default values after those without any, so that Python will know which arguments go into which values if you don't include all of them when calling the function:

```python
>>> def test(a, b=5, c, d=6):
...     print(a, b, c, d)
...
  File "<stdin>", line 1
SyntaxError: non-default argument follows default argument
>>> def test(a, c, b=5, d=6):
...     print(a, b, c, d)
...
>>> test(2, 3)
2 5 3 6
>>> test(2, 3, 10)
2 10 3 6
>>> test(2, 3, 10, 100)
2 10 3 100
```

Sometimes we might want to include some default arguments while excluding others. We do this by specifying which of the arguments we wish to pass a value to:

```python
>>> def test(a=0, b=0, c=0, d=0):
...     print(a, b, c, d)
...
>>> test(c=10)
0 0 10 0
>>> test(d=5, c=6)
0 0 6 5
```

2.2   Recursion

Recursion is the idea of a function calling itself. This can be useful in a wide variety of situations, but it can also be easy to misuse. As an example, see this simple case of recursion:

```
def test():
    test()
```

If this function is called, it will continue to call itself infinitely until Python prints out an error message and quits. This is because the function calls itself every time it is called – there is no case where the function ends without calling itself again. This case is called a base case, and it is important to include one in every recursive function. Here is a modified version of the above code that allows the initial call to specify a value limiting how many times the recursive call should be made:

```
def test(x):
    if x > 0:
        test(x - 1)
```

As an example of a more complex way of using recursion to solve a problem, here is an example of a recursive function that will sum a list where each element is either an integer or another list containing integers and lists:

```
def sum_lists(x):
    total = 0
    for i in x:
        if type(i) == int:
            total += i
        else:
            total += sum_lists(i)
    return total

data = [1, 2, [3, 4, [5, 6], 7], 8, 9]
print(sum_lists(data))
```

This program outputs 45, which is the sum of all the elements of the list, including all of the lists inside it. Notice that the base case in this function is if every element of the list passed to `sum_lists` is an `int`.

3   First-class Functions

In Python, functions are "first-class citizens". This means, essentially that they can be used in the same way as variables: they can be assigned to variables, passed to and returned from functions, and stored in data structures. As an example, here is a short program that stores functions inside of a list, then calls each one of those functions with the same argument.

```
def print_backwards(string):
    print(string[::-1])

def print_half(string):
    print(string[:len(string)//2])

```

```
7   def print_question(string):
8       print("Why do you want me to print '{}'?".format(string))
9
10  print_functions = [print, print_backwards, print_half, print_question]
11
12  for func in print_functions:
13      func("Hello, World!")
```

```
1   Hello, World!
2   !dlroW ,olleH
3   Hello,
4   Why do you want me to print 'Hello, World!'?
```

3.1 Map

A common operation in Python is to call the same function for every element in a list, then create a new list with the results. Previously, we would have accomplished this with a `for` loop:

```
1   def double(x):
2       return x * 2
3
4   data = [5, 23, 76]
5   new_data = []
6
7   for i in data:
8       new_data.append(double(i))
9
10  print(new_data)
```

```
1   [10, 46, 152]
```

If we use the `map` function, we can make this code much more simple:

```
1   def double(x):
2       return x * 2
3
4   data = [5, 23, 76]
5   data = list(map(double, data))
6   print(data)
```

```
1   [10, 46, 152]
```

The `map` function takes in a function and any iterable data type (such as a list, tuple, set, etc.) as arguments. It calls that function for every element of the other value passed, then returns a new data type containing all of those results. This new data type is a special map object, but in most cases we are fine with instantly converting it back to our initial data type (in the above example, a list).

3.2 Reduce

3.3 Lambda Functions

4   Exercises

> **Boilerplate**
>
> Remember that this lab *must* use the boilerplate syntax introduced in Lab 5, including the review exercises.

exercise.py

5 Submitting

We will be adding more exercises later. We have just not had the time to finish them. You will get an email about them.

Files to submit:

- exercise.py (Section 4)

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab8.zip` or `cse107_firstname_lastname_lab8.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have 7z installed. For Linux, look at lab 1 for instructions on how to create a tarball or use the "Archive Manager" graphical tool.

Upload your tarball or .zip file to Canvas.