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:

```
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:

```
>>> 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:

```
>>> 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

2.3 Lambda Functions

3 Iterables

3.1 Map

3.2 Reduce

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.