DS2000
Fall 2021
Handout: Writing Functions

All programming languages must be able to:

1. remember things,
2. repeat things,
3. communicate, and
4. make decisions.

Functions are all about #2: **repeating things**. Think of a function as a standalone mini-program: It takes some inputs, performs some operations, and generates a result. We define a function once, and we can execute the code as many times as we like -- repeating all those lines of code with a single invocation.

The code we've written in Python have already employed functions, because every program we write looks like this...

```
def main():
       print("Hello world!")

main()
```

At the tippy top, starting with the keyword *def*, we define a function called main. At the very bottom, where we have the name of the function followed by empty parentheses, we call the function.

Now we're doing more of the same thing!

Why Functions

Functions help us to organize our code. They save us from writing a big, long, ugly *main*. Functions are short and concise. Keep this rule of thumb in mind:

```
A function has ONE job.
```

We define a function to do its one little thing, and then we can execute that code many times, on all kinds of different inputs.

We define functions because they are:

1. ***Modular***. We end up with many small, single-purpose functions, which we can then piece together into a larger program. It's the same reason IKEA gives us a bunch of separate shelves that we assemble; it might be annoying, but it's much easier than if we had to build the whole bookshelf from scratch ourselves.
2. ***Testable***. We've written test cases for our Python programs in this class, which are great and useful. The drawback of what we've done so far, with everything in one big *main*, is that we need to eyeball the output to make sure it all worked as expected. Functions enable us to test our code

programmatically -- instead of a human eyeball, it's Python code that determines whether a function generated the correct output given an input. Much more reliable.

3. ***Reuseable***. Once a function is defined, we can use it in many programs, sometimes with wildly different purposes. As we know, Python provides a function that generates a random number, and I've used this function for programs that roll dice, deal a card from a deck, play a guessing game, play rock/scissors/paper, and basically any game of chance you can think of, because they're my favorite programs to write.

Name, Parameters, Return Type

A function is uniquely defined by three characteristics:

1. ***Name (required)*** . Similar to a variable name, a function is named so we can refer to it elsewhere in the program.
2. ***Parameters (zero or more).*** The data types of its expected input(s). A parameter is a variable that is attached to a function. A parameter is initialized by the function caller.
3. ***Return Type (zero or one).*** The data type of its expected output. A function performs some operation on its input, and returns the result to the function caller.

Here's an example of a simple function in Python, defined by the function writer:

```
def divide(x, y):
    ''' function: divide
        parameters: two numbers, x and y
        returns: a float, the result of dividing x / y
    '''
    result = x / y
    return result
```

Defining a simple function. (In real life, we probably don't need a function just to divide, but roll with it for now!)

Let's break it down to the important parts:

- **Name**: divide
- **Parameters**: two numbers, which could be floats or ints. The parameters are variables attached to the function, and they are named *x* and *y*. You can see that *x* and *y* are not initialized here in the function definition, but we know that we need them to be initialized so that we can use them -- they'll be given values by the function caller.
- **Return type**: a float. The function returns the result of *x / y*, which is a float even if *x* and *y* are both integers.

A few other things to notice here:

- There is a block comment written just under the function name. It should be in block comments so that pydoc will generate your function documentation correctly. For DS2000, we require both parameters and return type to be specified here, but including the name is optional (it's just something I like to do so I can see them all together in Spyder's highlights).
- The last line of the function is a **return** statement. Return is a reserve word in Python. Its job is to hand a value back to the caller. Once a return statement is executed, flow of control also returns to

the caller. If any additional code were written here underneath the return statement, it would never be executed.

Calling the function. Here's how it would look if I call the *divide* function from somewhere inside my main function:

```
def main():
    a = 14
    b = 5
    div = divide(a, b)
    print('The function returned:', div)

    div = divide(b, a)
    print('The function returned:', div)
main()
```

What to notice here:

- We had to know the *name, parameters*, and *return type* of this function in order to call it. That's how we know the correct syntax for invoking it.
- We've named our variables here *a* and *b* and initialized them to have the values 14 and 5. Therefore, 14 and 5 become the values associated with the function's parameters *x* and *y*. The values 14 and 5 are the **arguments** to the divide function.
- We call the function twice, which means we execute the code twice. But we get different results, because we switched the order! The first time, we get 14 / 5 = 2.8. The second time, we get 5 / 14 = .357.