DS2000
Fall 2021
Handout: Functions + Scope

From now on, all the code we write in DS2000 will be modular -- organized into functions that each has one job, is reusable, and is testable. Our code will be ***so great!!***

So now that we know how to write them... what happens when we call functions? That's our topic this time, and we're focused on parameters and arguments.

Parameter Names / Argument Names

Here's our simple function from the last handout, which simply divides two numbers.

```
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!)

This *divide* function has two parameters**.** A parameter is a variable that is attached to a function. It's still a variable, nothing special -- except that it exists only within the function.

- The labels of these variables are *x* and *y*.
- What are their values? We don't know!!!! The values are figured out when someone calls our function.

When we call the function from main, we provide **arguments,** which supply values for the function's parameters:

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

When we call our function, we provide arguments -- in this case, our arguments are a and b. The function parameter x will get the value 14 and y will get the value 5.

When we call the function, we specify the values of our parameters. Python matches up the order, so that...

- The parameter *x* will get the value 14
- The parameter *y* will get the value 5

- Then, boom, the *divide* function has two variables x and y, and it can move on with its computations just like it would with any other variables.

In the last handout, we called the function another way too, as in `div = divide(b, a)`. The function returns a different value this time, because:

- The parameter *x* will get the value 5
- The parameter *y* will get the value 14

Scope: Function variables don't exist outside the function
Our parameters *x* and *y* are variables attached to the function **divide**. But it has another variable, too -- *result*. All three of these (*x, y, divide)* are the function's local variables and they do not exist outside the function.

This is called **local scope.** It means that **everything in red below won't work.**

```
def divide(x, y):
    ''' function: divide
        parameters: two numbers, x and y
        returns: a float, the result of dividing x / y
    '''
    result = a / b  ### a and b don't exist outside main
    result = x / y
    return result

def main():
    a = 14
    b = 5
    div = divide(x, y)  ### x, y don't exist outside divide
    div = divide(a, b)
    print(result)  ### result doesn't exist outside divide
    print('The function returned:', div)
main()
```

Everything in red won't work, because function variables have local scope.

On the other hand, it means Python doesn't care if we reuse variable names in different functions. Each function is a little mini-program, and it has its own separate, independent variables. Even if they share names, Python considers them completely different from each other.

Because of **local scope**, we can reuse variable names and they are considered totally separate variables. This is fine:

```
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
```

```
def main():
    x = 14
    y = 5
    result = divide(y, x)
    print('The function returned:', result)
main()
```

Because of local scope, we can name our function variables the same thing. This program would print ***The function returned: .357*** *(note that we switched x and y when calling the function)*