Functions

Chapter 6

Functions

- A group of statements that exist within a program for the purpose of performing a specific task
- Can also be referred to as methods
- Sometimes referred to as divide and conquer
  - Takes a larger task and divides it up into smaller, easier tasks to accomplish
- A program that is divided up where each part has its own specific task is referred to as *modular*
- Python has built-in functions (such as print, format, input, etc.) but you're able to make your own as well

Function Benefits

- Simpler code
- Code reuse
- Better testing
- Faster development
- Team allocation

Function Naming

- Function names follow the same parameters as variable names (and are also case sensitive)
  - Uppercase and lowercase letters
  - Numbers (numbers cannot be the first character of the name)
  - Underscores
- The function *must* be defined with keyword “def”
- Everything that goes inside that function *must* be indented
  - Statements that are outside that function must go back to the original indentation
- Functions need a “function call” to execute

Function Anatomy

- Begin the function with keyword def
- Have a function name
- All statements you want inside the function are indented
  - Multiple statements can be allotted to one function
- Functions must have parentheses ()
- Functions must have a colon after the parentheses
- To execute (call) the function you must have the function name followed by parentheses

```
# sample function
def greetClass():
    print("Hello class")

def helloWorld():
    print("Hello world")
    print("Another statement")

# function calls
greetClass()
helloWorld()
```

Variable Passing

- You can have a variable in your main program that you use in a function
  - You need to “pass” the variable to that function
- Functions need to know they will be passed information
  - Whatever variables are used become a part of the function definition
  - Variables being passed to a function is a *parameter*
  - More than one parameter can be passed to a function as long as it’s in the definition
  - Once the function expects a parameter, you cannot call the function *without* the appropriate number of parameters

Parameters Anatomy

- You can pass items directly to a function or through variables
- The variable names you pass don't have to match the parameter names
  - The *order* you pass items 100% *does* have to match
- You *must* pass how ever many parameters are listed to not have Python error at the function call

```
# sample function with parameters
def greetClass(greeting):
    print(greeting)

# sample function
# more than one paramter
def sayHello(greeting, name):
    print(greeting, name)

# calls the functions
greetClass("Hello world")

firstName = "Kortni"
sayHello("Hey,", firstName)
```

Return Values

- You may want to use a variable from your function outside of your function
- Return values are needed to do so
- Need a return statement
- Return statements mark the end of the function
  - Nothing can occur in that function after it has returned a value

Return Statements Anatomy

- You can return a variable or a direct statement
- To use the return value from that function
  - Can be assigned to a variable
  - Can return directly into another statement

```
# sample function
# with return value
def getName():
    name = input("What is your name? ")
    return name

# other sample function
def returnName():
    return "Kortni"

# different ways to use values
name = getName()

print("Hello,", name)
print("Hello,", returnName())
```

Variable Scope

- Scope is where something is accessible in the program
- Variables have two classifications:
  - Local
    - Accessible only in the location they were declared
    - Declaring a variable inside a function makes that variable local to that function
  - Global
    - Accessible throughout the entire program
    - Functions can access variables that are not passed to them
    - Anything OUTSIDE the functions are global

Scope Anatomy

- Variables declared *outside* a function are global
- Local variables are declared *inside* a function
  - You cannot use a variable local to a function outside of that function unless you return it

```
# declares global variable
name = "Kortni"

# function using global variable
def greeting():
    print("Hello,", name)

# function with a local variable
def testFunction():
    dog = "Beagle"
    print(dog)
```

Global Downsides

- Global variable can commonly be referred to as bad programming practice
- They have a purpose in later materials
- Global variables can be hard to keep track of what is altering it
  - It is better to pass variables to functions
  - This forces the coder to pay attention to every access to that variable

Scope - Notes:

- A good way to avoid global variables is to put everything inside of a function
- Objects that don't have any specific use in a function can be placed inside a "main" function to act as a driver
- Do have to call your main function for it to work
  - This should be your only thing outside of the functions

```
# sample function with parameters
def greetClass(greeting):
    print(greeting)


# sample function
# more than one parameter
def sayHello(greeting, name):
    print(greeting, name)


# main function
# acts as a way to use the functions
def main():
    # calls the functions
    greetClass("Hello world")

    firstName = "Kortni"
    sayHello("Hey,", firstName)


# has to call main
main()
```

Function Notes:

- Functions *should not* be nested inside each other
- Functions *can* call one another
  - Good way to not use global variables
  - Can transfer variables from one function to another

```
# test function
def greeting():
    print("Hello")

# function that calls another function
def displayGreeting():
    greeting()

# calls other function
displayGreeting()
```

```
# example of what NOT to do
def greeting():
    print("Hello")

    def sayHello():
        print("This is a greeting")
```