Class 4: Functions and Loops

Date: 02/10/2019

Warm-Up

If you don't remember how to open up your text editor, look back to Worksheet 2! Or, just raise your hand.

When you're ready to start for the day, your first job is to make a program that asks the user to guess what Sally is selling, then uses an If/Then relationship to check if they guessed the right answer. The right answer should be "**`seashells`**"!

```
Do you know what Sally is selling by the seashore?
seashells
That's right!!
```

If the user types in anything other than "**`seashells`**," then this should happen in your program:

```
Do you know what Sally is selling by the seashore?
tacos
No, sorry…
```

Don't make two separate programs for this exercise! You want to create a single program that changes its outcome depending on what the user types in as their answer.

Functions: What are they?

At this point in the class, you've most likely started to realize that your code is becoming quite long and a bit unorganized. Today, we'll be learning a coding practice that will allow us to develop more organized and efficient code: functions. You can think of a **function** as a sort of mini-program, designed to accomplish some task, that is run within your larger, "main" program. Let's take a look at a quick example:

```
def main():
    print("Hey!")
main()
```

If you were to run it, the above function will simply print the following:

```
Hey!
```

Now, this seems like a roundabout way to make such a simple program! But, as it turns out, using this method in your programming will allow your code to become much more flexible and cleaner. Let's take a look at some important notes to understand how, exactly, this above program works:

- The function (mini-program) above that we are using is called `main()`. A function will always be followed by closed parentheses!
- The abbreviation "`def`" before the `main()` function stands for "define." This is where you create a "definition" for your function—in other words, this is where you write out the code for that particular mini-program. In our case, the code for our `main()` function is just a simple `print()` statement, that prints "`Hey!`" to the terminal!
- The computer doesn't actually run definitions, though, until you explicitly tell it to do so. That is, when you use the `def` qualifier before defining a function, you're essentially *teaching* the computer the definition of your function, NOT *instructing* the computer to run it (just yet).
- In order to actually execute this function (mini-program), we must **call** the function. To do a function call, all we have to do is write out the name of the function, followed by parentheses. That's it! In this particular program, the first call is a call to the `main()` function, found on the very last (third) line.

- Function calls can only be used *after the function is already defined!* When a computer runs a program, it reads from top to bottom. You can't instruct the computer to execute your function until you've taught the computer the definition of that function, first! It's good practice to keep all of your function definitions near the top of your program.

Take a look at this example program I put together that uses both functions and If/Then relationships! Remember that the computer will READ each section that starts with **`def`**, but won't run it until it comes across a *call* for that function. The first call is to the **`main()`** function, on the very bottom line.

```
def joke():
     print("How do you get a tissue to dance?")
     print("You put a little BOOGIE in it!")
def riddle():
     print("What has hands but doesn't clap?")
     print("A clock!")
def main():
     print("Would you like to hear a joke or a riddle?")
     answer = input()
     if ("joke" in answer):
          joke()
     elif ("riddle" in answer):
          riddle()
main()
```

Here's what the program will do when it is run!

```
Would you like to hear a joke or a riddle?
joke
How do you get a tissue to dance?
You put a little BOOGIE in it!
```

Believe it or not, but we've actually been secretly using functions since our *very first class*! Both **`print()`** and **`input()`** are two examples of **built-in** functions, which are functions defined by the Python language packages and libraries themselves. When we ask our computer to perform a "call" on one of these functions, the computer uses definitions that are pre-programmed in these libraries! When we make our own functions, we call them **user-defined**.

"For" Loops

Next, we'll look into another topic in computer programming: loops! Computers are especially convenient and useful for us because of their ability to perform complicated tasks in a very short amount of time. In particular, a computer can use **loops** to automatically and *quickly* perform any number of *repetitive* tasks!

Let's say I'd like to make a program that prints the word "banana" exactly five times. But, I don't want to write out 5 separate **`print()`** statements, each saying "banana." In a **`for`** loop, I can tell the computer that I would like a specific task to be repeated—in this case, to print out "banana." Let's write out the following code:

```
for x in range(5):
    print("banana")
```

Now try running the program:

```
banana
banana
banana
banana
banana
```

You are totally free to change the value in **`range(5)`** to any number that you'd like. You could print out 10 bananas, 100, 500, 2000, or more (That's a lot of bananas)!

******PLEASE be careful not to overwork your computer! When using loops, it's possible for your program to crash if you set your number of iterations TOO HIGH!******

Homework + Extra Learning

Your homework for the week will be to pull together everything you've learned so far, and take a shot at organizing your *Adventure* code using functions. It might help to create a new document, so that your original code is backed up!

Next-Week Snapshot: "While" Loops

Don't worry if you don't have the time to get to this step; we'll cover it next week!

When we want to run our code for a *specific number* of iterations, **`for`** loops are a great go-to. But, what if you want the loop to last indefinitely, that stops only when a certain condition is met? In cases like these, we'll use a **`while`** loop! Let's try this:

```
print("Do you want a banana?")

answer = input()

while("yes" in answer):

     print("You eat the banana.")

     print("Would you like another?")

     answer = input()
```

The above **`while`** loop will continue repeating forever while "**`yes`**" is found in the variable (container) **`answer`**. Next week, we'll review in more detail all that this program is doing, but in the meantime, try to figure out what is happening, yourself!

```
Do you want a banana?

yes

You eat the banana.

Would you like another?

yes

You eat the banana.

Would you like another?

no
```