3.11 Creating Fruitful Functions (CS 301 Extra)

Consider this function for printing a person's name by combining their first and last time into a string string:

```
def get_full_name(first, last):
    print(first + " " + last)
```

We trying calling and checking the result:

```
>>> result = get_full_name("Alice", "Anderson")
Alice Anderson
>>> print(result)
None
```

We see that `get_full_name` produces output ("Alice Anderson" is printed to the screen), but it is not a fruitful function (the `result` variable doesn't get a value, so we see `None` when we try to print it).

What if we want the output of a function to go to a variable, instead of to the screen?

One reason we might want to send the output of a function to a variable instead of just displaying the output on the screen is that maybe we want to do something additional with the value computed by the function before we display the result. For example, maybe we really want to print "Welcome Alice Anderson!" to the screen. We want a `get_full_name` function that helps us put together "Alice Anderson", but we want to add our own flourish ("Welcome" before the name and "!" after).

Turning `get_full_name` into a fruitful function so that we can save it's output in a variable is simple; we just replace `print` with `return`, like this:

```
def get_full_name(first, last):
    return(first + " " + last)
```

Now, if we rerun our early commands, we see the following:

```
>>> result = get_full_name("Alice", "Anderson")
>>> print(result)
Alice Anderson
```

Notice two things here: (1) calling `get_full_name` no longer causes something to be immediately displayed, and (2) `result` now contains Alice's full name, instead of just `None`. We have turned `get_full_name` into a fruitful function.

Now, we can produce the greeting we want:

```
full = get_full_name("Alice", "Anderson")
greeting = "Welcome " + full + "!"
print(greeting)
```

Getting this output:

```
>>> full = get_full_name("Alice", "Anderson")
>>> greeting = "Welcome " + full + "!"
>>> print(greeting)
Welcome Alice Anderson!
```

We have seen how print and return are similar, sending output either to the screen or to a variable, but there are some other important differences to remember.

Unlike when you print something, you don't need to put a return value in parentheses, so we could have written our function like this and it would have behaved the same:

```
def get_full_name(first, last):
    return first + " " + last
```

One very important different between printing and returning is that you can print many times, but once a return statement is encountered, the function stops immediately. Let's compare two version of a countdown function.

```
def countdown_print():
    print(3)
    print(2)
    print(1)

def countdown_return():
    return 3
    return 2
    return 1
```

Let's try calling both of them:

```
>>> countdown_print()
3
2
1
>>> result = countdown_return()
>>> print(result)
3
```

We see that all the print statements in `countdown_print` produce output, but only the first return statement in `countdown_return` did anything. As soon as `return 3` executed, the function finished executing, a return value of 3 was put in the result variable, and the `return 2` and `return 1` statements never ran.

If we really want the full countdown returned by `countdown_return`, we need to combine all the numbers in a single variable, and then return that with a single `return` statement, like this:

```
def countdown_return():
    value = '3'
    # this represents the character you get when you press Enter on your keyboard
    value += '\n'
    value += '2'
    value += '\n'
    value += '1'
    return value
```

And now we get what we want when we call it:

```
>>> result = countdown_return()
>>> print(result)
3
2
1
```