BYU IDAHO CSE 110 | Programming Building Blocks

13 Prepare: Preparation Material

Overview

Have you noticed in this course, or in other aspects of your life, that there are some tasks you want to do over and over again? Or perhaps there are common steps that you need to follow, but the result might be slightly different depending on the context?

In programming, we have the ability to encapsulate chunks of reusable code into a *function* that we can refer to whenever we want to accomplish that task. This can range from simply performing a few steps in order, to computing complicated results to return back to another part of the program.

Functions are very powerful, because they capitalize on the idea of abstraction. That is, that you can focus really intently on a certain portion of the code, but then once that's done, you don't have to worry about the details any longer, instead, you can just refer to it by name, and know that the proper steps will be taken care of.

The next course in the programming sequence, CSE 111 *Programming with Functions*, will spend the entire semester focusing on the nuances of functions, writing them, testing them, using functions from others, etc. Similarly, the course after that, CSE 210 *Programming with Classes*, will dive deeper into this idea of leveraging abstractions in your programs to create complex solutions. Especially for programs that you would have a hard time keeping all of the pieces in your brain at once; and creating

them in a way that makes them easy to update when new features are needed.

In this lesson, you'll learn the basics of functions that can start you on the path toward writing larger and more involved programs.

Preparation Material

Watch the following videos:

- Introducing Functions (10 mins)
- Demo: Functions (8 mins)
- Parameterized Functions (6 mins)
- Demo: Parameterized Functions (5 mins)

I. DEFINING FUNCTIONS

As shown, you *define* a function by using the `def` keyword and following your function with parentheses `()` and a colon `:` as follows:

```
def your_function_name():
    # code here
    # for the
    # body of the function
```

II. CALLING FUNCTIONS

Later in your program, you can call the function by giving its name and including the parentheses afterward:

```
your_function_name()
```

The following code demonstrates a function that displays text on the screen. Then, it is called twice later in the program:

```
# First define the function
def print_message():
    print("Hello CSE 110 World!")

# Call the function now
print_message()

# Call it again here
print_message()
```

This produces the following output:

```
Hello CSE 110 World!
Hello CSE 110 World!
```

III. PASSING PARAMETERS TO FUNCTIONS

When you define a function, you can declare parameters, which are values that it receives from the place that called it. For example, consider a function **`print_double`** that receives a number, doubles it, and prints the result on the screen:

```
def print_double(value):
    double_value = value * 2
    print(double_value)
```

Then, you can call this function and pass it any value:

```
print_double(12) # outputs 24
print_double(3) # outputs 6
print_double(42) # outputs 84
```

IV. RETURNING VALUES

Extending the previous example, suppose you want to have the program calculate the double value and return it, instead of printing it to the screen, in case you want to use it in further calculations. In this case, you can use the **`return`** statement:

```
def get_double(value):
    double_value = value * 2
    return double_value
```

Now the value is given back to the code that called it:

```
new_number = get_double(4)
```

The variable **`new_number`** would now have the value 8.

V. VARIABLE SCOPE

Please be aware that variable names are only valid for the function they are defined in. And, you can even use the name variable name in different functions for different values. This can get a little complicated and you'll see it in much more detail in future classes. But at this point, you need to be aware that whether you use the same variable name or not, the function will have it's own copy of the value. And you can't use it outside of the function.

```
def get_double(value):
    double_value = value * 2
    return double_value

new_number = get_double(4)

# ERROR: This does not work, because the variable "double_value" is only alive during
# the body of the function. Down here, we have chosen to give the return value the name
print(double_value) # BAD CODE
```

The same concept holds true for parameters:

```
def print_message(the_message):
    print(the_message)

# it works just fine to use the same variable name as the function did...
the_message = "Message 1"
print_message(the_message)

# but it also works to use a different variable...
user_message = "Message 2"
print_message(user_message)
```

In either of the two cases above, when you are in the function, you'll want to use the variable name **`the_message`** regardless of what it was called down below.

If this is still a little confusing, that's ok, that's why there is a whole additional course to explore these nuances.

Style

As you start to think about names for your functions and your variables, you'll find that most often variables refer to *things* so it is most common to use nouns for them (for example, price, name, response, etc.). Functions on the other hand *do* things or perform actions, so it is more common to use verbs for their names (for example, print_message, get_initials, open_file, display_error).