Get Dressed

Imagine a friend invites you to go see a movie. You slept late, so you reply, 'Meet you there. I just need to get dressed.'

Your friend knows what you mean by 'get dressed,' but doesn't know the details, like what shirt you'll put on.

Now think about the details of everything you did to get dressed today: deciding what to wear, getting clothes out of a dresser or a closet, putting on each piece of clothing.

The words "get dressed" seem so simple, but when you think about every detail that's involved in getting dressing, you understand those two words represent a lot of complexity.

Using a phrase like "get dressed" gives us a way to talk and think about the process of getting dressed without needing to think about every specific detail.

People do this all the time. Phrases like "go to the dentist," "make a sandwich," or "meet for coffee" describe activities that would be very tedious to talk about step by detailed step.

The idea of taking something complex and defining a simpler way to refer to it is called ***abstraction***. Abstraction is a fundamental idea in coding. It's how programmers keep from getting overwhelmed by details and complexity.

In this conversation, you'll learn how to define and use ***handlers***, one of the most fundamental ways to define abstraction in code.

Take a few minutes to think about other simple phrases that represent much more complicated activities or ideas.

Next, learn how programmers also use simple phrases to refer to more complicated operations.

Calling a handler

You probably didn't realize it at the time, but you've already taken advantage of abstraction and handlers when you used `print()` to print text to the console. In Python, `print()` is a handler. When you use it, you are ***calling*** the handler:

```
print('Hello, world!')
print(360)
```

Just as you perform many activities when you get dressed, many things happen when you call the `print()` handler, including:

- Turning whatever parameter you hand in to it, including numbers, into a bunch of characters.
- Adding a "newline character" to the end of the bunch of characters, so each call to `print()` ends up on a new line.
- Making that bunch of characters show up in the console.

In this case, you're calling a handler that someone else has already defined. You don't need to know every detail about how `print()` works in order to call it, i.e., you don't have to know the steps required to define `print()` in order to use it in your code.

This is a large part of what makes handlers so powerful. They provide a way to combine detailed steps into a definition that can be used again and again.

For the rest of this assignment, you'll practice calling handlers and learn how to define handlers of your own.

Repeating Yourself

You've written a lot of code in this class. Recall that in repl.it, the code you write "plays" from top to bottom, and you see the outcome on the right-hand side of the window.

You've also learned how to print to the console. The following code is going to print a nursery rhyme.

```
print('Row, row, row your boat')
print('Gently down the stream')
print('Merrily, merrily, merrily, merrily')
print('Life is but a dream')
print('             ~        ')
print('Row, row, row your boat')
print('Gently down the stream')
print('If you see a crocodile')
print('Dont forget to scream')
print('             ~        ')
print('Row, row, row your boat')
print('Gently down the stream')
print('This song is quite repetitive')
print('Can you spot the theme')
```

Here is a silly joke that is really true:

There are two things that programmers try to avoid:

Repeating themselves
Repeating themselves

Writing the same code more than once gives you more things to write, to understand and to fix.

Here are some questions just to think about (they don't require you to formally respond):

How many of the lines of code above are repeated? How can you fix this?

You will find out answers to these questions, and more, in the next section.

A Single Piece of Work

You can "wrap up" code that you may want to use more than once in a handler. You can define a handler using the Python verb `def`.

Because handlers can contain multiple lines of code, the definition is a little more complicated. Here is a very simple handler definition:

```
def rowTheBoat():
    print('Row, row, row your boat')
    print('Gently down the stream')
```

The ***custom Python verb*** being defined is `rowTheBoat`. You will learn more about the `()` later on, but for now remember that any word followed by parentheses is a handler (even if you didn't define it yourself).

The code indented below the first line is called the ***body*** of the handler.

These lines of code are different from others you've seen in class. They display nothing in the console. That's because ***defining*** a handler only describes what the handler would do if it ever played.

To actually play the code, you have to ***call*** the handler. Typing the custom Python verb that the handler defines will call the handler. Here is an example:

```
rowTheBoat()
```

When you've called a handler in a repl.it project, you'll see results appear on the right-hand side of the window. Calling the `rowTheBoat()` handler will print the nursery rhyme in the console area.

Do this:

In repl.it, create a new project called **nurseryRhyme**. In this project, define and call the handler described in this section, `rowTheBoat()`.

Create another, new project in repl.it called **myRhyme**. In this project write your own handler that prints a rhyme of your choice to the console, then call your new handler.

Pay attention to the example above so that you know where all of the parentheses `()` and other grammatical elements go!

Finally, let's review what you've learned.

Wrapup

You've now seen how handlers can be used to group code together so that a complicated task can be performed with a single line of code, even if that line of code relies on many lines written somewhere else.

We have also talked a lot about some new vocabulary.

We'll continue this conversation when I return...

Do this:

To this Schoology assignment, submit the links to the two repl.it projects I asked you to create.