BYU IDAHO CSE 110 | Programming Building Blocks

08 Prepare: Preparation Material

Overview

In this lesson, you will continue working with loops and learning how to solve more complex problems.

Preparation Material

In the previous lesson, you learned about **`while`** loops. In this lesson, you will build on that foundation to learn about another kind of loop called a **`for`** loop.

Watch the following videos which discuss both kinds of loops:

- Loops (5 mins)
- Demo: Loops (3 mins)

As shown, there are a few different ways you can tell the computer to loop through code over and over again.

I. LOOPING THROUGH A LIST

One example of looping, is to loop, or iterate, through each item of a collection or a list, one at a time. You will learn about lists in more detail in future lessons, but the following shows an example of looping through a list of words to print them on the screen one at a time:

```
items = ["crayon", "scissors", "paper", "glitter glue", "markers", "pens"]

for item in items:
    print(f"The item is: {item}")
```

The output of this code is:

```
The item is: crayon
The item is: scissors
The item is: paper
The item is: glitter glue
The item is: markers
The item is: pens
```

Just as with **`if`** statements, the colon **`:`** at the end of your **`for`** statement tells the computer that there is a block of code coming. Then, you indent all the code that is in the block. The code that is indented is the code that will be run on each item.

When looping through each item, you can do much more with it than simply printing it out. You could include an **`if`** statement, to check if it meets certain conditions, and then use it in a math expression, or anything else that you've learned how to do in your programs.

> **Frequently Asked Questions:**
>
> What about the variable **`item`**? Is that a special name? Where does it get defined?
>
> There is nothing special about the name **`item`** in this case. It could have just as easily been called **`thing`** or **`writing_instrument`**.
>
> The for loop syntax here is very clever. When you type **`for item in items:`**, the list variable, **`items`** must already exist, but the variable for each individual element of the list, in this case **`item`**, is defined right in that statement. In short, it's saying, "Go through each element of this list and assign a new variable **`item`** to be the value each time through.

II. LOOPING THROUGH NUMBERS

It's very common in programming to want to loop through a series of numbers. You could create a list of numbers, just like the list of words in the last example, like this:

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

But this could get really long if you wanted to work with a large amount of numbers, so there is an easier way, using **`range`**.

You can create the same list of numbers like this:

```
numbers = range(10)
```

This creates a list of numbers for you that starts at **`0`** and goes up until (but not including) **`10`**.

You could assign this list to a variable like **`numbers`** as in the example above, and then loop through each number in the list of numbers, but it is more common to just put the **`range`** code right in your for loop like this:

```
for number in range(10):
    print(number)
```

The output of this code is:

```
0
1
2
3
4
5
6
7
8
9
```

Hint from Instructor:

In programming languages we usually start counting at 0 instead of 1. There are reasons for this, but more than anything, it's just something you'll have to get use to.

The **range** function gives you other options like starting at a different number, or counting by 3's, or even going backwards. In all of these cases, the idea of looping through the values is the same. You start with the first one and then continue on, one by one, through the list.

To start with a different number, you provide two values to the **range** function. If you provide two numbers, it will start with the first one and continue up until (but not including) the second one.

If you provide three values to the **range** function, it will use the third number as the "step size" or in other words the number to count by.

```
# This loop will start with 100, and go up to, but not including 200
for i in range(100, 200):
    print(i)

# This loop will do the same thing, but this time, it will count by 10's
for i in range(100, 200, 10):
    print(i)
```

III. LOOPS WITHIN LOOPS

A loop will blindly repeat any code that is in its body. That code could include if statement, mathematics, new variable definitions, or anything else—including other loops!

The following example shows displays the numbers 0-4:

```
for i in range(5):
    print(i)
```

Output:

```
0
2
3
4
```

We can add a second, inner loop, that for each one of those displays the numbers 10-12 (with two `-` characters in front to make it easier to visualize).

```
for i in range(5):
    print(i)
    for j in range(10, 13):
        print(f"--{j}")
```

Output:

```
0
--10
--11
--12
1
--10
--11
--12
2
--10
--11
--12
3
--10
--11
--12
4
--10
--11
--12
```

Notice how the inner loop is run every time the outer loop displays it's number. Inner loops can be very helpful in iterating through a two-dimensional structure, such as the pixels in an image.

> **Hint from Instructor:**
>
> While we generally prefer variable names that are more descriptive than a single letter, if the variable will only be used for counting in a simple loop it is considered standard to use `i`. Then, if you have an inner loop, you use `j`, and a third inner loop would be `k`. If you have more than three loops you should consider if there is a simpler way to accomplish your task, and if there really isn't, you should at least move to more descriptive variable names at that point.