BYU IDAHO CSE 110 | Programming Building Blocks

09 Prepare: Preparation Material

Overview

In most problems we encounter, we have lists or collections of things: stock prices over time, students in a preschool class, medical lab results for multiple patients, photo shoots that we have scheduled, etc.

Whenever we have a list of items, it's common to want to add or remove from the list, find things in it, or apply some action to some or all of the items in the list. To solve these problems, we have lots of tools and strategies to work with lists in programs. In this lesson, you'll learn the basics of how to store and manipulate lists. Then, in future lessons, you'll continue to work with them and learn techniques to solve problems with them.

One way that you'll use this in upcoming lessons is to load in a file containing data and need to analyze it in some way. For example, you might load a file containing vaccination rates for a disease in various countries in the world and then do some analysis to find out what the lowest or highest rate is, what the average rate is, or how many countries have rates below a certain threshold. The concepts you are learning this week are building you up to be able to answer these questions in these meaningful cases.

Preparation Material

Hint from Instructor:

To this point, most of the videos we've used for preparation material have come from the Microsoft series on beginning programming with Python. While they do

have some that cover the kinds of topics we'll be discussing for the next few lessons, we're going to deviate from them because they don't get as detailed in the fundamental concepts that we need and instead go deeper into other areas that we won't be discussing this semester.

So if you see videos in their YouTube channel discussing collections, etc., it wouldn't be a problem to watch them, but they won't be our standard preparation material.

Watch the following videos:

- Lists in Python (7 mins)
- Lists of Numbers (7 mins)

I. CREATING LISTS

There are different kinds of collections in Python and other programming languages, but the most fundamental one in Python is the *List*. A list is simply a collection of variables, and you can put any kind of variable in a list. To declare a list, you use the square brackets `[]`. If you are creating an empty list (to put items in later), you can declare it as follows:

```
movies = []
```

If you want to start the list with items in it, you can put them inside the square brackets with commas in between like this:

```
movies = ["Toy Story", "Pride and Prejudice", "Star Wars"]
```

In the last example, we put strings into the list, but you could put numbers in the list as well:

```
quiz_scores = [89.2, 78.5, 92.4, 93.8]
```

Hint from Instructor:

Technically, you can mix types of variables and have a list that contains different types of variables (e.g., strings and numbers), but it often makes it harder to work with the list later, so that is generally discouraged. Instead, make a list that has a clear purpose and put only those things in that belong. Then, if needed, make another list for other things.

Naming List Variables

Because lists contain many items, its common to give them variable names that are plural. For example, **`quiz_scores`** is preferred to **`quiz_score`** because there will be more than one. The computer doesn't care about this, but it will make it much easier for you (and others that may use your program) to keep track of everything.

While we have used the square brackets `[]` to create our lists, you could have written this out in a longer form using the keyword **`list()`**, such as: **`movies = list()`** . The square brackets are a shorthand way to type that, and are the preferred method. Because **`list`** is a keyword, you should avoid using it as the name of your list variable:

```
# Avoid using list as your variable name like this:
list = [] # BAD EXAMPLE
```

If you do name your variable **`list`**, you'll see that the editor will change its color because it thinks you are referring to the keyword. Instead, choose a variable name that is more descriptive of the kind of items that will be in the list.

II. ADDING ITEMS TO A LIST

Once you have a list variables, you can add items to it using the **`append`** function. Notice that when you use it, you'll need to use

parentheses `()` afterward, just like when we have called other functions like **`print`** and **`input`**. You'll learn more about calling functions and even creating your own in the next course.

You can add items to the end of the list, or *append* them, as shown:

```
books = []

books.append("1 Nephi")
books.append("2 Nephi")
books.append("Jacob")
books.append("Enos")
```

III. ITERATING THROUGH EACH ITEM IN A LIST

In previous lessons, you've already learned how to iterate through the items of a list using the "for each" syntax in Python:

```
print("Your books are:")

for book in books:
    print(book)
```

IV. WORKING WITH LISTS OF NUMBERS

When you have a list of numbers, a common task is to want to find the sum of all of these numbers or the largest number, or the smallest, etc. One strategy to accomplish this is to declare a variable outside a loop that you can reference in the loop as you iterate through each individual item.

For example, the following code declares a variable **`running_total`** outside the list and then as it iterates through the numbers, it adds each to the running total:

```
receipts = [12.24, 18.50, 4.99, 21.75]

running_total = 0
```

```
for receipt in receipts:
    running_total = running_total + receipt

# Display the total
print(f"The total is: {running_total:.2f}") # This displays: The total is: 57.48
```

A new operator

As you saw in the previous example, we can add to a variable with code like this:

```
age = 30
age = age + 1

print(age) # This displays: 31
```

But because this is so common, there is an operator that can do this in one step. This is done using the `+=` operator:

```
age = 30
age += 1

print(age) # This displays: 31
```

These two versions do the same thing, but people generally prefer to use the `+=` operator because it's simpler, easy to understand, and less prone to mistakes.

With this in mind, the previous code example could be updated to be the following:

```
for receipt in receipts:
    # Note the use of the += operator here
    running_total += receipt
```

V. STYLE

As you may have noticed in these examples, it is considered good style to put blank lines in your program to separate parts of the code that perform different types of tasks. With this in mind, you should consider blank lines before and after your loops, as well as after declaring your list variables.