Learning Activity (3 of 3): Looping through Strings

Preparation Material

Looping Through Strings

You can iterate through each letter of a string using either style of **`for`**-loop (the "for each" style or the "for i in range" style). For example, you can loop through each letter one at a time with the following code:

```
first_name = "Brigham"
for letter in first_name:
    print(f"The letter is: {letter}")
```

The output of this code is:

```
The letter is: B
The letter is: r
The letter is: i
The letter is: g
The letter is: h
The letter is: a
The letter is: m
```

Just as before, there is nothing special about the variable name **`letter`**, any name could have been used, but the choice of **`letter`** helps us keep the code nice and easy to read.

Accessing Letters by Position

Sometimes you need to access a letter by its position (or index) in a string. In other words, you might want to know the third or seventh letter in a string. This will be useful when comparing letters at the same place in two strings, such is in your project for this lesson.

You can access a specific letter in a string by using the square brackets `[]`, such as `word[3]` or `word[12]`. But be aware that that the count starts at 0, so you will use `3` to get the *fourth* letter (not the third letter). The following example gets the fourth letter of the name:

```
first_name = "Brigham"

fourth_letter = first_name[3] # Notice, using 3 instead of 4
print(fourth_letter) # outputs a 'g' on the screen
```

Iterating through each letter using an index

In the previous sections you learned how to iterate through each letter in a string, but this only gave you access to the letter but not the index of that letter. In many cases, this is sufficient. However, in other cases you need to know both the letter and its index. (For example, in your project you need to know the letter and also check its position in another word.)

In this case, rather than looping through letters, you can loop through the index numbers. If you knew the length of the word, you could do this as follows:

```
word = "book"
number_of_letters = 4

for index in range(number_of_letters):
    print(index)
```

This code will output the following:

```
0
1
2
3
```

Then, you can use the square brackets to access the letter at that index as follows:

```
word = "book"
number_of_letters = 4

for index in range(number_of_letters):
    letter = word[index]
    print(f"Index: {index} Letter: {letter}")
```

This code will output the following:

```
Index: 0 Letter: b
Index: 1 Letter: o
Index: 2 Letter: o
Index: 3 Letter: k
```

With this example, you have access to both the letter and the index at each step of the loop.

Finding the String Length

The previous example assumed the number of letters, or *length* of the string, would always be 4, but this will not work if the string has more or fewer letters. Instead of typing the 4 directly, you can let the computer find the length by using the **`len`** function. The following code shows how to use **`len`** to get the length of a string:

```
dog_name = input("What is your dog's name? ")
letter_count = len(dog_name)

print(f"Your dog's name has {letter_count} letters")
```

The output of this code could be:

```
What is your dog's name? Rover
Your dog's name has 5 letters
```

Combining the **`len`** function with the earlier loop is very powerful, because now you can iterate through the indexes and letters of strings of any length as follows:

```
word = "book"
number_of_letters = len(word) # Notice this can now work for any string

for index in range(number_of_letters):
    letter = word[index]
    print(f"Index: {index} Letter: {letter}")
```

This code will output the following:

```
Index: 0 Letter: b
Index: 1 Letter: o
Index: 2 Letter: o
Index: 3 Letter: k
```

Lists

Many of the examples you see with **`for`** loops often iterate through a list of items, including numbers in a list, or letters in a string. This naturally extends to working with other properties of lists, such as finding the length of the word or the position of the letter.

The material from this week introduces some of these concepts, but please be aware that the material for next week focuses entirely on working with lists. So, while you'll start to use lists a little bit now, you will get much more practice and learn the details even better as you continue on to next week.

▶ (Optional) Python Enumerate

Activity Instructions

Write a program that asks the user for their favorite letter. Then, loop through a programmed word one letter at a time. If the letter in the programmed word is the user's favorite, you'll first print it out as a capital letter, then later you will "hide" it. If the letter is not their favorite you will print it out as a lower case letter.

Printing without new lines

To this point, whenever we have used a print statement, it has always been on it's own line, so that the next line starts on a new line. If you *do not* want the print statement to end with a new line, you can specify the **`end`** as follows:

```
print("This is line one.", end="")
print("This is line two.")
```

This outputs the following:

```
This is line one.This is line two.
```

Notice, that because you told the first print statement to end with nothing (by using `""`), it does not end with a newline and the next line prints directly following it.

Working in Steps

The final version of this program will hide letters by turning them into an underscore _. To help you get to that point, you will work through three steps.

Step 1: Make the Favorite Letter uppercase

Create a string variable to hold the word `"Commitment"`.

Write code that loops through the word letter by letter. If the letter is `"m"`, print it as a capital letter. If the letter is anything else, print it out as a lowercase letter.

For this part, it is ok to print each letter on it's own line.

The output should look as follows:

```
c
o
M
M
i
t
M
e
n
t
```

Step 2: Clean up the Output

Change the print statements so that each letter is not printed on its own line, but rather they are printed out next to each other on the same line.

Also, change the program to allow the user to specify the letter (rather than always using `"m"`). Make sure your program matches the letter in the word, regardless of whether the user entered it as a capital or lowercase, and regardless of whether that letter was a capital or lowercase in the original word.

The output should look as follows:

```
What is your favorite letter? T
commiTmenT
```

Step 3: Hide the Letter

Change the program, so that instead of capitalizing the user's favorite letter, it "hides" it, and replaces it with an underscore in the display.

The output should look as follows:

```
What is your favorite letter? m
co__it_ent
```

Or another example could be:

```
What is your favorite letter? t
commi_men_
```

Sample Solution

When your program is finished, please view a sample solution of this program to compare your approach to that one.

You should work to complete this checkpoint program first, without looking at the sample solution. However, if you have worked on it for at least an hour and are still having problems, you may feel free to use the sample solution to help you finish your program.

- Sample solution

Testing Procedure

Verify that your program works by making changes and ensuring that it contains to work as expected:

1. Try a few different letters as the favorite.
2. Try a different word.
3. Try a word with a space in it.
4. Try a letter that is not present in the word.

Submission

You have now completed all of the learning activities for the week!

Make sure to:

- Return to Canvas and submit the associated quiz.

Up Next

- Check Your Understanding

Other Links:

- Return to: Week Overview | Course Home