Chapter 6 - Strings

Strings

Lab - Strings

Lab 1

Counting Uppercase and Lowercase Characters

You are going to write a program that takes a string and prints out two messages. One message tells you how many uppercase characters are in the string. The other messages says how many lowercase characters are in the string. The program will ignore all numbers and special characters (punctuation, symbols, etc.).

String Methods

You will need two string methods that were not covered earlier to help with this project:
* `.isupper()` - Returns `True` if the character is uppercase, `False` if the character is lowercase
* `.islower()` - Returns `True` if the character is lowercase, `False` if the character is uppercase

Variables

You will need three variables for this project. One variable will count all of the lowercase variables, another to count the uppercase variables, and a string variable.

```
lower_count = 0
upper_count = 0
my_string = "Roses are Red, Violets are Blue"
```

Iterating Over the String

The next thing to do is iterate over the string. A simple for loop works best.

```
for char in my_string:
```

Checking for Uppercase and Lowercase

It does not matter if you check for an uppercase character first or check for a lowercase character. Let’s start with lowercase characters. Ask if the character is lowercase and increment the appropriate counting variable.

```
for char in my_string:
    if char.islower():
        lower_count += 1
```

What you **do not** want to do is use an `else` statement. This will not give you an accurate count. Asking if a special character is lowercase will return `False`. However, that does not mean they uppercase characters. Special characters are neither uppercase or lowercase. So use an `elif` and ask if the character is uppercase. If so, increment the uppercase counting variable.

```
    elif char.isupper():
        upper_count += 1
```

Print the Results

The final step is to print the messages with the count values. We are going to use the `format` method for string interpolation.

```python
print("There are {} lowercase characters.".format(lower_count))
print("There are {} uppercase characters.".format(upper_count))
```

There should be 4 uppercase characters and 21 lowercase characters.

▼ **Code**

```python
lower_count = 0
upper_count = 0
my_string = "Roses are Red, Violets are Blue"

for char in my_string:
    if char.islower():
        lower_count += 1
    elif char.isupper():
        upper_count += 1

print("There are {} lowercase characters.".format(lower_count))
print("There are {} uppercase characters.".format(upper_count))
```

Lab 2

Reverse a String

You are going to write a program that takes a string and prints it in reverse order.

Variables

You are going to need two variables. The first is the original string. Since strings are immutable, we will need a second variable to represent the reversed string. Make the reversed string an empty string.

```
my_string = "The brown dog jumps over the lazy fox"
reversed_string = ""
```

Since the string needs to be reversed, our loop should start at the end of the string and work its way to the first character. So the for loop used in Lab 1 will not work. We are going to use a while loop. To do this, we need a variable that represents the index. Normally, the index starts at zero, but this loop will go backwards through the string. The starting index needs to be the last character in the string. The `length` method returns the length of a string. However, the length of a string is always one greater than the last index (because indexes start counting at zero). So the starting index should be the length of the string minus one.

```
index = len(my_string) - 1
```

String Iteration

The while loop should run as long as `index` is greater or equal to 0.

```
while index >= 0:
```

Reversing a string comes down to taking the a character from the end and putting it at the front of a new string. This will be done by appending the character at `index` to `reversed_string`. It is important to remember that strings are immutable. The line of code below is overwriting `reversed_string` with a new string. It is not updating the contents of the string.

```
    reversed_string += my_string[index]
```

Decrement the `index` variable to avoid an infinite loop.

```
    index -= 1
```

Printing the result

Once the loop has finished running, print `reversed_string`.

```
print(reversed_string)
```

You should see `xof yzal eht revo spmuj god nworb ehT`.

▼ **Code**

```
my_string = "The brown dog jumps over the lazy fox"
reversed_string = ""
index = len(my_string) - 1

while index >= 0:
    reversed_string += my_string[index]
    index -= 1

print(reversed_string)
```

Lab 3

Swapping the Case of Characters

You are going to write a program that takes a string and prints a new string where all of the uppercase letters become lowercase, and the lowercase letters become uppercase.

Variables

You are going to need two string variables. The first one represents the original string, and the second variable represents the modified string. The modified string can be an empty string.

```
original_string = "THE BROWN DOG JUMPS over the lazy fox"
modified_string = ""
```

String Iteration

It does not matter if you start at the beginning of the string or the end for iteration. A simple for loop is the easiest way to iterate through the `original_string`.

```
for char in original_string:
```

String Methods

You are going to use the `isupper` and `islower` methods to test if a character is uppercase or lowercase. In addition, you will be using the `upper` and `lower` methods to convert characters to its new case.

Conditional

We will test if a character is lowercase. It does not matter if you decide to test for uppercase, just be sure to make the appropriate conversion.

```
    if char.islower():
```

If this is true, then append the uppercase version of the character to the variable `modified_string`.

```
        modified_string += char.upper()
```

If the conditional is false, then append the lowercase version of the character to `modified_string`.

```
    else:
        modified_string += char.lower()
```

You do not need to worry about special characters. Converting them to uppercase or lowercase has no effect.

Printing the Results

Once the loop has finished, print both the original string and the modified string. Use the `format` method to add some context to the output.

```
print("The original string is: {}".format(original_string))
print("The modified string is: {}".format(modified_string))
```

You should see the following output:

```
The original string is: THE BROWN DOG JUMPS over the lazy fox
The modified string is: the brown dog jumps OVER THE LAZY FOX
```

▼ **Code**

```python
original_string = "THE BROWN DOG JUMPS over the lazy fox"
modified_string = ""

for char in original_string:
    if char.islower():
        modified_string += char.upper()
    else:
        modified_string += char.lower()

print("The original string is: {}".format(original_string))
print("The modified string is: {}".format(modified_string))
```

Lab 4

Count the Vowels

You are going to write a program that counts the number of vowels that appear in a string. For the purpose of this exercise, vowels are `a`, `e`, `i`, `o`, `u`.

Variables

For this project, you will need three variables. One will be the string. Another will be a string of all the vowels (`"aeiou"`). The final variable will be a count of all the vowels.

```
my_string = "The Brown Dog Jumps Over The Lazy Fox"
vowels = "aeiou"
count = 0
```

String Iteration

A simple for loop will check each character in the string.

```
for char in my_string:
```

Checking for a Vowel

You could have a long set of conditionals that ask if `char` is `a`, then ask if it is `e`, etc. Since the vowels are represented as a string, much more efficient way to do the same thing is to ask if `char` is one of the characters in `vowels`. This is done with the `in` operator.

```
    if char in vowels:
```

There is one problem with the line of code above. The string `vowels` is a string of lowercase characters. If `char` is an uppercase character, the conditional would return `False`. The easiest way to solve this issue is convert `char` to lowercase.

```
    if char.lower() in vowels:
```

Incrementing the Counter

When `char` is found in the string of vowels, increment the `count` variable.

```
        count += 1
```

Printing the Result

The last step is to print the result. Use the `format` method to provide some context. It is possible that a string could have only one vowel. The text you print should be grammatically correct. Use a conditional to determine if there is one vowel. Use the appropriate grammar if true, or use a different sentence if there are more than one vowel. Having zero values would use the same syntax as many vowels.

```
if count == 1:
    print("There is 1 vowel in the string")
else:
    print("There are {} vowels in the string".format(count))
```

You should see that there are 9 vowels in the string.

▼ **Code**

```python
my_string = "The Brown Dog Jumps Over The Lazy Fox"
vowels = "aeiou"
count = 0

for char in my_string:
    if char.lower() in vowels:
        count += 1

if count == 1:
    print("There is 1 vowel in the string")
else:
    print("There are {} vowels in the string".format(count))
```

Lab Challenge

Replacing Vowels with a *

You are going to write a program that takes a string called `my_string` and returns the string but with a `*` in the place of vowels. Assume that vowels are `a, e, i, o, u`. For example, if `my_string = "Hello"`, then your program will print `"H*ll*"`.

Important, the variable `my_string` is already declared as an empty string. Add a value to it and test your code. However, **do not** submit your code to be graded with the variable declaration. The auto-grader will declare the variable for you.

Code Visualizer