Lab 4: While loops; strings as sequences

Labs are graded for participation rather than correctness. Keep all your lab code in your course GitHub repo to receive credit for your work. We'll be looking to see that you have at least partially completed all problems in each lab.

If you finish the lab assignment early, you may get started on the homework.

Getting started

In your local repo, create a folder for Lab 4. Create new .py files for this week's lab problems and save them in the Lab 4 folder.

Documentation and format guidelines

The guidelines from previous labs still apply. No new guidelines this week!

Problem 1: Logarithm

Write a program that asks the user to enter a positive integer, *n*, which is a power of 2, and **uses a while loop** to calculate the logarithm base 2, of *n*. For this assignment, you can assume the user gives you good input.

> About Logarithm: The *log* function is the inverse of the exponent function. Suppose you calculate the power of one number raised to another: $z = x^y$. Then we can say that the log base *x* of *z* is *y*. For example, $4^3 = 64$. And therefore, $\log_4 64 = 3$.
>
> In computer science, we typically use base-2 when calculating logs. We use the notation lg to mean "logarithm base 2." We can compute lg *n* as some *x* such that $2^x = n$. Shortcut/cheat-sheet:
>
> - lg(1) = 0
> - lg(2) = 1
> - lg(4) = 2
> - lg(8) = 3
> - lg(16) = 4
> - lg(32) = 5

A loop is a good mechanism for calculating a logarithm. The user is inputting a power of 2, and so the question we really need answered is how many times can we *divide* their input by 2 until we get to the number 1, which is as small as we can get? For example, if the user enters 8:

- 8 / 2 = 4 # 1 division
- 4 / 2 = 2 # 2 divisions
- 2 / 2 = 1 # 3 divisions

... and done! lg(8) is 3 because we could divide in half 3 times.

Example input / output # 1 (user input is shown in blue)

```
Enter a positive power of 2: 1
lg(1) = 0
```

Example input / output # 2 (user input is shown in blue)

```
Enter a positive power of 2: 2
lg(2) = 1
```

Example input / output # 3 (user input is shown in blue)

```
Enter a positive power of 2: 4
lg(4) = 2
```

Example input / output # 4 (user input is shown in blue)

```
Enter a positive power of 2: 1024
lg(1024) = 10
```

Optional extension: refactor your function to calculate the logarithm of a number to any base e.g. log base 3 of powers of 3, or log base 10 of powers of 10.

Problem 2: Factorial

Write a program that **uses a while loop** to calculate *n!,* the product of n and all the non-negative, non-zero integers below it. For example, 4! is 4 * 3 * 2 * 1 = 24

Problem 3: Calculator

Write a program that performs basic arithmetic.

Step 1: Prompt the user to enter a number.

Step 2: Ask the user to enter the next step in the calculation they would like to perform. Accept input in the following format:
<arithmetic operator> <value>
… where **<arithmetic operator>** is +, -, *, or /, and value is the positive integer they would like to use the operator with.

Step 3: Perform the requested operation, keeping track of the subtotal. For example, if the user entered “10” at the first prompt, then entered “* 2”, the subtotal would be 10 * 2 = 20.

Use a while loop to repeat steps 2 and 3 until the user enters “q” or “Q” to quit. Print the current subtotal at the end of each iteration then print the final total after the user has quit. Don’t worry about order of operations in this problem—execute the calculation as soon as it’s entered. Don’t worry about rounding errors either!

Example input / output (user input is shown in blue)

```
Enter a number: 10
Enter the next step in the calculation: + 2.4
Subtotal = 12.4
Enter the next step in the calculation: - 10
Subtotal = 2.4
Enter the next step in the calculation: * -3
Subtotal = -7.199999999999999
Enter the next step in the calculation: q
Total = -7.199999999999999
```