PER SCHOLAS

ALAB 308.3.1: Practical Loops

Version 1.0, 10/12/23
Click here to open in a separate window.

Introduction

This assignment will have you create several different types of loops according to a specification. This will showcase the power of loops and iteration alongside JavaScript operators, comparators, and conditional statements for data creation, processing, and manipulation.

Objectives

- Use for loops to iterate a specific number of times.
- Use for of loops to iterate through iterable data-like strings.
- Use while loops to iterate based on a condition.
- Use the break and continue statements to control loop flow.

Submission

Submit your completed lab using the **Start Assignment** button on the assignment page in Canvas.

Your submission should include:

- A GitHub link to your completed project repository.

Instructions

Initialize a new git repository in a local project folder, and create a JavaScript file to contain your code. Complete the activities below. For each of the activities, decide whether it is best to use for, for of, or while loops to implement the solution.

Commit frequently! Every time something works, you should commit it. Remember, you can always go back to a previous commit if something breaks.

Part 1: Fizz Buzz

To begin, solve the following classic “Fizz Buzz” problem. There are a few different ways to do this - experiment with what you think is the most efficient. Once you have solved the problem, ask yourself if there could be another way; and if so, would it be better.

Accomplish the following:

- Loop through all numbers from 1 to 100.
- If a number is divisible by 3, log “Fizz.”
- If a number is divisible by 5, log “Buzz.”
- If a number is divisible by both 3 and 5, log “Fizz Buzz.”
- If a number is not divisible by either 3 or 5, log the number.

Remember to commit your solution once it is working.

Part 2: Prime Time

Now we will move onto something slightly more complex.

Context: A prime number is any whole number greater than 1 that cannot be exactly divided by any whole number other than itself and 1. For example, the number 5 is prime because it cannot be divided by 4, 3, or 2; it can only be divided by itself (5) and 1. Similarly, the numbers 7 and 11 are prime. As numbers become larger, determining whether or not they are prime is increasingly difficult, but loops make this process relatively easy!

Write a script that accomplishes the following:

- Declare an arbitrary number, `n`.
- Create a loop that searches for the next prime number, starting at `n` and incrementing from there.
- As soon as you find the prime number, log that number and exit the loop.

Continuing with the example above, if `n` is equal to 4, your loop should log 5. Similarly, if `n` is 5, it should log 7, and if `n` is 9, it should log 11. Test your loop with higher numbers and reference an online prime number table to determine the accuracy of your code.

Be careful! If you set `n` to a number too large, your loop could take a long time to process.

Part 3: Feeling Loopy

As a final task, solve the following practical problem regarding string processing.

Context: A CSV file, or “Comma-Separated Values” file is traditionally used to store tabular data. You may be familiar with CSVs through past use of programs such as Microsoft Excel or Google Sheets. While each of these programs save their data in different formats to preserve style (e.g., font color or cell backgrounds), at their core, they are storing CSV data.

CSV data looks like this:

```
ID,Name,Occupation,Age\n42,Bruce,Knight,41\n57,Bob,Fry Cook,19\n63,Blaine,Quiz Master,58\n98,Bill,Doctor’s Assistant,26
```

Not very nice to look at. The “\n” is an escaped Line Feed, which indicates that the following data should begin on a new line, as follows:

```
ID,Name,Occupation,Age
42,Bruce,Knight,41
57,Bob,Fry Cook,19
63,Blaine,Quiz Master,58
98,Bill,Doctor’s Assistant,26
```

As you may have guessed, these values being “comma-separated” means that each comma is also a delimiter. This is why this type of data is traditionally viewed in tables. Here is how the data looks once fully parsed:

ID	Name	Occupation	Age
42	Bruce	Knight	41
57	Bob	Fry Cook	19
63	Blaine	Quiz Master	58
98	Bill	Doctor’s Assistant	26

Your task is to write a script that accomplishes the following:

- Loop through the characters of a given CSV string.
- Store each “cell” of data in a variable.
- When you encounter a comma, move to the next cell.
- When you encounter the “\r\n” sequence, move to the next “row.”
- Log each row of data.
  - You do not need to format the data, the following works well.
  - `console.log(cell1, cell2, cell3, cell4);`

You can make the following assumptions:

- There will only be 4 cells per row.
- There will be no escaped characters other than “\n”.

Use the example string provided above to test your program. Once you are confident it is working correctly, try the following string to see if your program works properly with other data.

```
Index,Mass (kg),Spring 1 (m),Spring 2
(m)\n1,0.00,0.050,0.050\n2,0.49,0.066,0.066\n3,0.98,0.087,0.080\n4,1.47,0.116,
0.108\n5,1.96,0.142,0.138\n6,2.45,0.166,0.158\n7,2.94,0.193,0.174\n8,3.43,0.20
4,0.192\n9,3.92,0.226,0.205\n10,4.41,0.238,0.232
```