**CSCI 136 Assignment – Benford**

**January 11th, 2023**

~~**Due Date: Sunday, 1/15/2023 11:59PM**~~

**Due Date: Thursday, 1/19/2023 11:59PM**

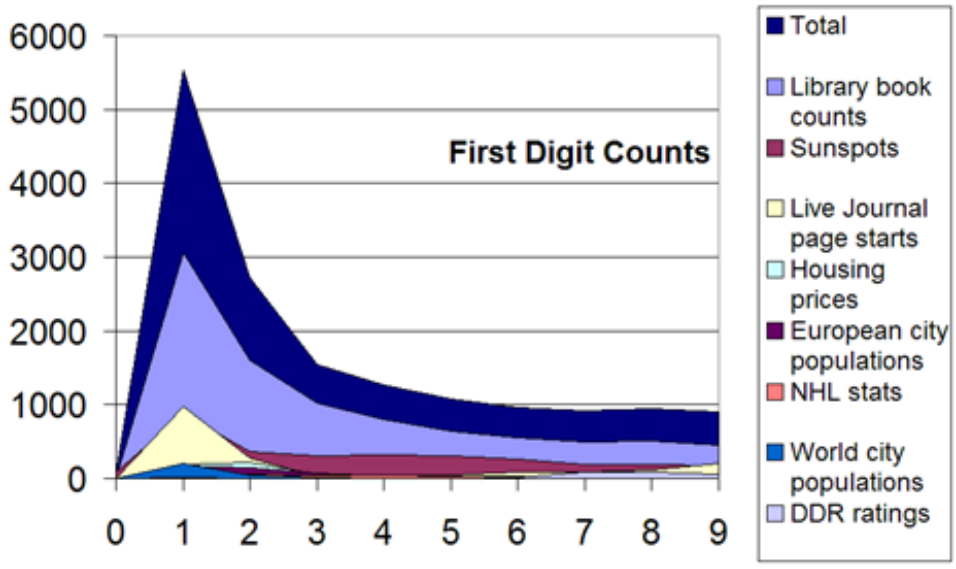
**Points: 40**

**Topic: File Input, String Processing, Benford's Rule**

The natural world is full of hidden and beautiful mathematics. The whorls of a conch shell hide the Fibonacci sequence and its Golden Ratio, plants grow in fractal patterns, and comets trace hyperbolic patterns through the solar system. All those beautiful patterns hide in the data of human observation.

So, how are the populations of every town in Quebec and the number of posts by various authors to a hockey bulletin board related?

A good explanation of why your intuition may or may not be correct can be found in Benford's Rule: https://plus.maths.org/content/os/issue9/features/benford/index .




**Submission:**

In this assignment, you will write a program that determines the distribution of initial digits in a set of data. It will count how many 0's, are in the leftmost position (position 0), how many 1's, how many 2's, and so on. Your program will take two command line arguments - the position n of the digit you are counting and the name of the file containing the numbers to use. The program first reads from the file a number x and then a list of x numbers and then outputs a list of 10 values: the frequency that each digit 0–9 appears as the nth digit of one of the input numbers.

After following all instructions below, please submit your Python script (Benford.py) and a writeup file (Benford.docx) in Moodle. Upload these as individual files – do not zip them. Be sure the (main) file has a header section that includes the file name, your name, credits, a description of the file, the class and other useful information. Make sure your class and all methods are thoroughly commented.

- `Benford.py`
- `Benford.docx`

**Grading:**

This assignment is worth 40 points. You will be graded according to the following criteria:

| | **Points** |
|---:|---:|
| Program Compiles and Runs | 2 |
| Header Comment | 2 |
| Comments on all Methods | 2 |
| Implemented countDigits | 1 |
| Implemented nthDigitBack | 1 |
| Implemented nthDigit | 1 |
| Implemented nthDigitTally1 | 1 |
| Implemented nthDigitTally | 1 |
| Implemented readMysteriousNumbers | 1 |
| Implemented main | 2 |
| countDigits Runs Correctly | 2 |
| nthDigitBack Runs Correctly | 2 |
| nthDigit Runs Correctly | 2 |
| nthDigitTally1 Runs Correctly | 2 |
| nthDigitTally Runs Correctly | 2 |
| readMysteriousNumbers Runs Correctly | 2 |
| Program Produces Correct Output | 4 |
| Write Up of Results on All Data Sets | 10 |
| **Total** | **40** |

You can use the included `TestBenford.py` script to test your code. Download this file and run it, as follows:

```
> python TestBenford.py
```

from the (Anaconda) command prompt. The results will show if your code works and may point to areas where you need to do more work. The `TestBenford.py` script assumes you have named your program and classes as specified, and that all methods are implemented according to the following specification.

**Benford.py**

Name your script `Benford.py`. and implement the following class. See below for detailed descriptions of what each method should do. It is important that your code implements the class as specified. Test your methods as you write them.

```
class Benford
-----------------------------------------------------------------

# Initialize members of the class.
__init__()

# Returns the number of digits in the number num.
int countDigits(int num)

# Returns the digit at the nth position from the right
# in the number num.
int nthDigitBack(int n, int num)

# Return the digit at the nth position from the left
# in the number num.
Int nthDigit(int n, int num)

# Updates and returns the tally list with a new count
# based on the corresponding digit n in the number num.
int [] nthDigitTally1(int n, int num, int [] tally)

# Returns a tally list of all digits in the nth position.
int [] nthDigitTally(int n, int [] nums)

# Opens a file and reads all the numbers into a list.
# Assumes the first number is the count of subsequent numbers
# in the file and does not include that number in the list.
int [] readMysteriousNumbers(String fName)
```

Write the method `countDigits()`.

`countDigits(int num)` calculates and returns the number of digits in the integer `num`. `countDigits` should return 1 for numbers in the range of 0–9, 2 for numbers in the range of 10–99, 3 for 100–999, etc. Hint: you can use repeated division by 10 to calculate the number of digits in num. There are other approaches.

Write the method `nthDigitBack`.

`nthDigitBack(int n, int num)` finds and returns the nth lowest order digit in num, i.e., the nth digit from the right. We take the rightmost digit to be the 0th digit. `nthDigitBack` should return -1 for digits beyond the "start" of the number. This is a flag that we can use that value to determine that there was no digit in a particular position. Hint: you can use repeated division by 10 again, followed by the modulo operator to extract the rightmost digit. There are other approaches.

```
nthDigitBack(0,123) ⇒ 3
nthDigitBack(1,123) ⇒ 2
nthDigitBack(2,123) ⇒ 1
nthDigitBack(3,123) ⇒ -1
nthDigitBack(0,0) ⇒ 0
nthDigitBack(3,18023) ⇒ 8
```

Write the method `nthDigit` using the methods `nthDigitBack` and `countDigits`.

`nthDigit(int n, int num)` extracts and returns the `n`th highest order digit of `num`, i.e., the `n`th digit from the *left*. We take the leftmost digit to be the 0th. `nthDigit` should evaluate to -1 for digits beyond the "end" of the number. For example:

```
nthDigit(0,123) ⇒ 1
nthDigit(1,123) ⇒ 2
nthDigit(2,123) ⇒ 3
nthDigit(3,123) ⇒ -1
nthDigit(0,0) ⇒ 0
nthDigit(3,18023) ⇒ 2
```

Write the method `nthDigitTally1`, using `nthDigit`.

`nthDigitTally1(int n, int num, int [] tally)` assumes that tally is a 10-element list tallying the number of `n`th digits seen so far. It updates and returns tally to reflect the `n`th digit of num. In other words, if d is the `n`th digit of num, then increment the dth element of tally. Examples:

```
nthDigitTally1(2, 1072, [0,0,1,2,0,0,3,0,9,0]) ⇒ [0,0,1,2,0,0,3,1,9,0]
nthDigitTally1(0, 2541, [0,0,1,2,0,0,3,1,9,0]) ⇒ [0,0,2,2,0,0,3,1,9,0]
```

Write the method `nthDigitTally`, using `nthDigitTally1`.

`nthDigitTally(int n, int [] nums)` returns the list, `tally`, of frequencies of 0–9 as the `n`th digits of all the numbers in the list `nums`.

Here's a sample test case. Following are enrollments in Research Triangle Park colleges and universities from Fall 2000 (Credit: "Research Triangle Regional Partnership" website http://www.researchtriangle.org/data/enrollment.html , link no longer works).

| Institution | Enrollment |
|---|---|
| Duke University | 12176 |
| North Carolina Central University | 5476 |
| Louisburg College (Junior College) | 543 |
| Campbell University | 3490 |
| University of North Carolina at Chapel Hill | 24892 |
| North Carolina State University | 28619 |
| Meredith College | 2595 |
| Peace College | 603 |
| Shaw University | 2527 |
| St. Augustine's College | 1465 |
| Southeastern Baptist Theological Seminary | 1858 |

Assume the variable `enrollments` contains the enrollment numbers from that table. Then:

```
nthDigitTally(0, enrollments) ⇒ [0,3,4,1,0,2,1,0,0,0]
```

Write the method `readMysteriousNumbers`.

`readMysteriousNumbers(String fName)` takes a file name as an input parameter and reads integers from that file. It returns a list of the numbers suitable for input to `nthDigitTally`. The first entry in the file will be a count of the number of entries in the list.

Here is the university enrollment data from above:

```
11
12176
5476
543
3490
24892
28619
2595
603
2527
1465
1858
```

For this data, `readMysteriousNumbers` should produce the list of integers:

`[12176,5476,543,3490,24892,28619,2595,603,2527,1465,1858].`

In the data files used for these exercises, the first number indicates how many data entries there are, and the numbers that follow are the data entries, one per line. The TestData.txt file contains the university enrollment data shown above.

Write a `main` function that runs only if the script is executed directly.

Finally, compose a test `main` function. This function should be defined inside `Benford.py` but not inside the `Benford` class. It should be defined so that it will only be executed if the `Benford.py` script is executed stand-alone, not imported into another script. (See separate slides and article about Defining Main Functions for guidance on this.)

Your main function should read a number `n` followed by the name of the file holding the data from command line arguments. The program should tally the `n`th digits of the numbers in the data set from the file and print out the results as shown below. For example, executing your program from the command line as:

```
> python Benford.py 0 TestData.txt
```

The output should be:

```
0s: 0
1s: 3
2s: 4
3s: 1
4s: 0
5s: 2
6s: 1
7s: 0
8s: 0
9s: 0
```

Once your program has run correctly on TestData.txt, run it on the additional data files: LibraryBooks.txt, LiveJournal.txt, and SunSpots.txt, all of which are included with this assignment.

**Benford.docx**

Using no more than 2 pages, write and submit a summary MS-Word document named Benford.docx. Include the following in this document:

- Write up your results on the four data files: `LibraryBooks.txt`, `LiveJournal.txt`, and `SunSpots.txt`.
    - Include the tallies of digits for each file.
    - Do your results appear consistent with the Benford Rule?
- Find a data source on the web and transform it into a format suitable for input to readMysteriousNumbers. The data must all be separate measurements of a single type of phenomenon, similar to the data in the assignment files. The numbers should be non-negative integers. The data set should have at least 250 measurements in the list, but more would be better. Create a file to include this data with the format used for the included data files. Write up the results of running your Benford.py script on this data.
    - Cite the source of your data.
    - Briefly describe what the data is.
    - Show the tallies of digits for this data.
    - Do your results appear consistent with the Benford Rule?

**Helpful hints:**

Use the example data file that was provided. Use it to develop and test your code thoroughly before moving on to the larger data files.

Do not assume that because your code works on one data file that it will work on others. Test and debug thoroughly.

Work incrementally. Build and test incrementally, starting with easier aspects. Add functionality incrementally, testing with each change.

Review the examples from our book for text file input closely. Choose an approach that makes sense and works for you. There are different ways to accomplish the task at hand. Some are easier than others.

Review the slides and article about Defining Main Functions. Make sure you understand what is called for with the main function for this lab and how to make it work correctly in the several situations in which it will be used.

After following all instructions above, please submit your Python script (Benford.py) and a writeup file (Benford.docx) in Moodle. Upload these as individual files – do not zip them. Be sure the (main) file has a header section that includes the file name, your name, credits, a description of the file, the class and other useful information. Make sure your class and all methods are thoroughly commented.

- `Benford.py`
- `Benford.docx`

## EXTENSIONS

You will not submit the answers to these extensions as part of your assignment submission. You should work through and understand them to learn more and practice. You may be held responsible for the content within, so you really should work through them. With that said, this are outside the scope of what will be submitted for this lab. Do not change your original submittal to address these extensions.

From me:

1.

Was the inclusion of the number of data elements as the first line of the file necessary in your implementation? With other, older languages, it is necessary to know how much space to allocate for storage of the data. What in Python alleviates the need for this?

2.

Separate from your submission for this lab assignment, consider other ways you could define a class to do the tallies of digits similar to what was done in this lab assignment. What would you do differently and what would you do the same?

3.

One different way to approach this problem is the recognize that the data files are text files and the data is initially read in as strings. Could you define a class to work directly on the strings rather than convert the numbers as a whole to integers? What assumptions would you have to make to do this? Would the code be shorter or longer than the code you submitted for the lab assignment? Would the code be more efficient? I.e., would it run faster?