NYU Tandon School of Engineering

CS-UY 1114 Spring 2023

Homework 07

Due: 11:59pm, April 6th, 2023

Submission instructions

1. You should submit your homework on **Gradescope**.
2. For this assignment you should turn in 3 separate `.py` files named according to the following pattern: `hw7_q1.py`, `hw7_q2.py`, etc.
3. Each Python file you submit should contain a header comment block as follows:

```
"""
Author: [Your name here]
Assignment / Part: HW7 - Q1 (etc.)
Date due: 2023-04-06, 11:59pm
I pledge that I have completed this assignment without
collaborating with anyone else, in conformance with the
NYU School of Engineering Policies and Procedures on
Academic Misconduct.
"""
```

No late submissions will be accepted.

REMINDER: *Do not use any Python structures that we have not learned in class.*

For this specific assignment, you may use everything we have learned up to, **and including**, variables, types, mathematical and boolean expressions, user IO (i.e. `print()` and `input()`), number systems, and the `math` / `random` modules, selection statements (i.e. `if`, `elif`, `else`), and `for`- and `while`-loops. Please reach out to us if you're at all unsure about any instruction or whether a Python structure is or is not allowed.

Do **not** use, for example, dictionaries, lists, tuples, and/or object-oriented programming.

Failure to abide by any of these instructions will make your submission subject to point deductions.

Problems

1. **Read Between The Words (`hw7_q1.py`)**
2. **The Root Of This Problem Is That It's Average (`hw7_q2.py`)**
3. **Damage Report (`hw7_q3.py`)**

Problem 1: *Read Between The Words*

Write a function `get_word_count()` that will accept the address of a `txt` file (a string) and will returns the number of words contained in this file. Any one of these files may look like the following (let's call it `voltaire.txt`):

```
Italy had a Renaissance, and Germany had a Reformation, but France had
Voltaire; he was for his country both Renaissance and Reformation, and half the
Revolution.
He was first and best in his time in his conception and writing of history, in
the grace of his poetry, in the charm and wit of his prose, in the range of his
thought and his influence.
His spirit moved like a flame over the continent and the century, and stirs a
million souls in every generation.
```

The following code:

```
def main():
    count = get_word_count("voltaire.txt")  # assumes txt file is in same
directory
    print(f"This file has {count} word(s).")

main()
```

Would thus print out:

```
This file has 84 word(s).
```

Here are some things to keep in mind:

1. You may not assume that this file exists. If it doesn't, your program should print a warning message to the user of your choice and then return 0.
2. You can ignore punctuation here. That is, `"Hello, World!"` contains 2 words.
3. You may *not* use the `split()` method in this problem, as it results in the creation of a `list` object, which we have not covered yet.
4. You may assume that the last character of every line in the file will either be a non-whitespace character or a newline character `'/n'`.

Hint: The title of this problem.

Problem 2: *The Root Of This Problem Is That It's Average*

Write a function `get_root_of_average()` that will accept a string parameter representing a `txt` file containing a series of numbers, such as the following (let's call this file `numbers.txt`):

```
1 2 3 NULL 5 6.03
```

Your function will also accept an integer parameter, `root`. What it will then do is calculate and return the `root`-root of the *average* of all the numbers in the file.

```

```

```
def main():
    cubed_root = get_root_of_average("numbers.txt", 3)
    print(cubed_root)

main()
```

Output:

```
WARNING: Could not cast 'NULL' into a float.
1.5036945962049748
```

Sanity check:

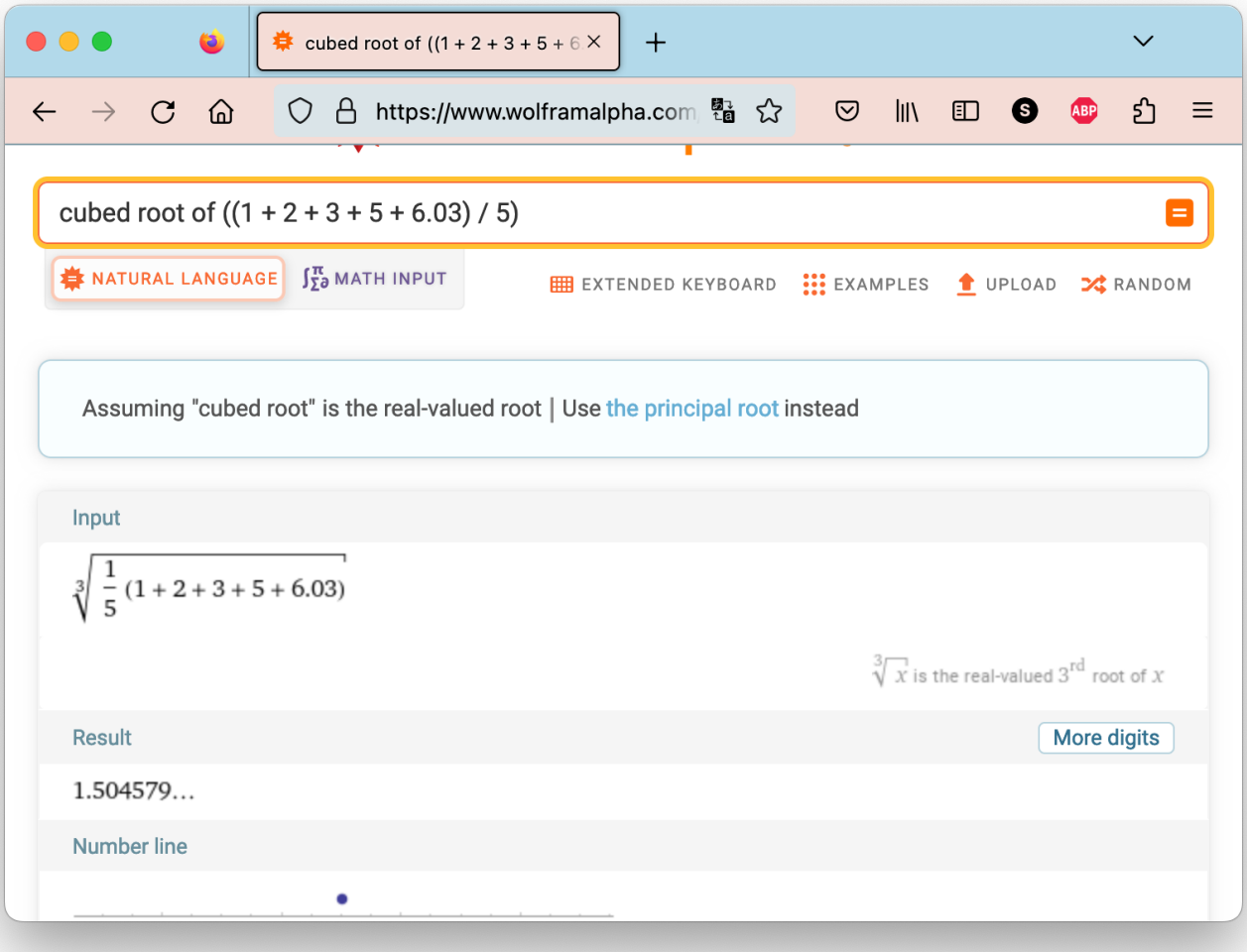

Figure 1: Using **WolframAlpha** to confirm the result. Because of floating-point precision, your result may vary a little, like it does with mine.

Here are some things to keep in mind:

1. You may not assume that this file exists. If it doesn't, your program should print a warning message to the user of your choice and then return 0.
2. If the user chooses not to enter a root argument, your function should default to the square root.

3. If your program runs into a non-numerical value, print a warning message (like the one above) and continue onto the next number. Do *not* count it as an occurance of a number.
4. You may *not* use the `split()` method in this problem, as it results in the creation of a `list` object, which we have not covered yet.
5. You may assume that the file only has one line, but that line may not contain any actual numbers. Watch our for the potential of dividing by zero when calculating the average in this case. If it happens, simply return 0.

Problem 3: *Damage Report*

Let's say we are a computational biologist, and we have this (potentially very large) DNA sequence that we want to use for analysis. As is the case with most data, this strain has a few components that are missing or corrupted.

You are given two things:

- **A DNA Sequence** (*str*): A sequence of nucleotides (`'A'`, `'C'`, `'T'`, and `'G'`) whose integrity may be compromised by corrupted characters.
- **The Name Of A File** (*str*): This file contains a series of numbers, one on each line. These numbers represent *indices* where the DNA sequence is expected to be corrupted.

Your goal here is not to actually fix the DNA sequence at the locations that the file contains, but rather to make sure that the locations *within* the file are valid. A valid location/index is one that:

1. Is an integer.
2. Is an integer smaller than the length of the DNA sequence.

What we're going to do is create an **error** report for each one of the lines in the indices file. For example, let's say our sequence is:

```
"ACTGC AXT"
```

And our indices file looks like this:

```
5
20
7.0
```

This file is telling us that at locations 5, 20, and 7.0, there is a corruption in the DNA sequence. The two problems with this file are that:

- `20` is out of range.
- `7.0` cannot be casted into an `int`.

Your function, `create_error_log()` would accept the DNA sequence and the name of the indices file, and generate a report (i.e. a `txt` file called `error_log.txt`) of which indices are not valid. Your report must look like this:

```
```

```
INDEXERROR at LINE 2: The value read, 20, is larger than the sequence length of
9.
VALUEERROR at LINE 3: The value read, '7.0', cannot be casted into an integer
to be used as an index.
```

This is a small example using a very short file. Your function must work for a file containing any number of lines. As usual, you may not assume that the file referenced by the second parameter exists. If this ends up being the case, your error log should simply include the line:

```
FILENOTFOUNDERROR: The file specified was not found or could not be opened.
```

A sample call to this function might look like this:

```
def main():
    create_error_log("ACTGC AXT", 'indices.txt')

main()
```