Lab Exercise #4

Assignment Overview

This lab exercise provides practice with file processing.

You will work with a partner on this exercise during your lab session. Two people should work at one computer. Occasionally switch the person who is typing. Talk to each other about what you are doing and why so that both of you understand each step.

Mirmir testing: create one file named `lab04.py` and keep adding your functions to the same file.

Part A: Leap Year

A leap year in the Gregorian calendar system is a year that is divisible by 400 or a year that is divisible by 4 but not by 100. Write a function `named leap_year` that takes one string parameter. It returns `True` if the string represents a leap year, and returns `False` otherwise.

For example, 1896, 1904, and 2000 are leap years, but 1900 is not. Therefore,

```
leap_year('1896')
```

returns `True`.
(Optional challenge: write the function suite as one line.)

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab04.py") for grading via the Mirmir system.**

Part B: Rotate

Write a function `rotate(s,n)` that has one string parameter `s` followed by a positive integer parameter `n`. It returns a rotated string such that the last `n` characters have been moved to the beginning. If the string is empty or a single character, the function should simply return the string unchanged. Assume that `n` is less than or equal to the length of `s` and that `n` is a positive intger.

For example:
`rotate('abcdefgh',3)` returns `'fghabcde'`
(Optional challenge: write the function to handle `n` larger than the length of `s`.)

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab04.py") for grading via the Mirmir system.**

Part C: Digit Count

Write a function named `digit_count` that takes one parameter that is a number (int or float) and returns a count of even digits, a count of odd digits, and a count of zeros that are to the left of the decimal point. Return the three counts in that order: `even_count, odd_count, zero_count`. Be careful of the "edge case" where the number starts with a decimal point—conversion of such a

number to a string places a zero before the decimal point. See correct behavior in the final test case below.

For example:
`digit_count(1234567890123)` returns `(5, 7, 1)`
`digit_count(123400.345)` returns `(2, 2, 2)`
`print(digit_count(123.))` returns `(1, 2, 0)`
`print(digit_count(.123))` returns `(0, 0, 0)`

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab04.py") for grading via the Mirmir system.**

Part D: Float Check

String has a method `s.isdigit()` that returns `True` if string `s` contains only digits and `False` otherwise, i.e. `s` is a string that represents an integer. Write a function named `float_check` that takes one parameter that is a string and returns `True` if the string represents a `float` and `False` otherwise. For the purpose of this function we define a `float` to be a string of digits that has at most one decimal point. Note that under this definition an integer argument will return `True`. Remember "edge cases" such as "45." or ".45"; both should return `True`.

For example:
`float_check('1234')` returns `True`
`float_check('123.45')` returns `True`
`float_check('123.45.67')` returns `False`
`float_check('34e46')` returns `False`
`float_check('.45')` returns `True`
`float_check('45.')` returns `True`
`float_check('45..')` returns `False`

(Optional challenge: write this function suite in one line.)

★ **Demonstrate your completed program to your TA. On-line students should submit the completed program (named "lab04.py") for grading via the Mirmir system.**