

CMPSC-132: Programming and Computation II

Homework 1 (100 points)

Due date: September 10th, 2021, 11:59 PM EST

Goal: Revise your Python concepts and using them to solve problems using Python's built-in data types

General instructions:

- The work in this assignment must be your own original work and be completed alone.
- The instructor and course assistants are available on Teams and with office hours to answer any questions you may have. You may also share testing code on Teams.
- A doctest is provided to ensure basic functionality and may not be representative of the full range of test cases we will be checking. Further testing is your responsibility.
- Debugging code is also your responsibility.
- You may submit more than once before the deadline; only the latest submission will be graded.

Assignment-specific instructions:

- Download the starter code file from Canvas. Do not change the function names or given starter code in your script.
- Each question has different requirements, read them carefully and ask questions if you need clarification. No credit is given for code that does not follow directions.
- If you are unable to complete a function, use the pass statement to avoid syntax errors

Submission format:

- Submit your HW1.py file to the Homework 1 Gradescope assignment before the due date.
- As a reminder, code submitted with syntax errors does not receive credit, please run your file before submitting.

Section 1: Overview of functions

| Points | Name | Input type | Output type |
|--------|----------------------------|------------|-------------|
| 15 | rectangle(perimeter, area) | int, int | int/bool |
| 20 | frequency(aString) | str | str, dict |
| 20 | successors(file) | str | dict |
| 15 | getPosition(num, digit) | int, int | int/bool |
| 15 | hailstone(num) | list | list |
| 15 | largeFactor(num) | int | int |

Section 2: Function details

rectangle(perimeter, area)

Returns the longest side of the rectangle with given perimeter and area if and only if both sides are integer lengths. In a rectangle $perimeter = 2w + 2h$ and $area = w * h$

Hints:

- The built-in [round\(number\)](#) function rounds a number to the nearest integer.
- To do integer division (also known as [floor division](#)), use the // operator. Floor division, as opposed to true division, discards any fractional result from the output.
- the built-in [float.is_integer\(\)](#) method returns True if the float instance is finite with integral value, and False otherwise

Preconditions:

| Inputs | | |
|--------|------------------|---|
| int | <i>perimeter</i> | The perimeter of the rectangle expressed as an integer. |
| int | <i>area</i> | The area of the rectangle expressed as an integer. |

| Outputs | |
|---------|--|
| int | The longest side length for the rectangle that meets the given requirements. |
| False | False is returned if no rectangle exists that meets the requirements. |

Examples:

```
>>> rectangle(14, 10) # Represents a 2x5 rectangle
5                      # 5 is the larger of the two sides
>>> rectangle(25, 25) # Represents a 2.5x10 rectangle
False                 # 2.5 is not an int
```

frequency(aString)

Returns a string that describes the most frequent item in aString ('vowel' or 'consonant') and a dictionary with the frequency count of each alphabet letter (in lowercase) in the aString. When a vowel and a consonant have the same frequency, it returns the description of the first letter with that frequency in aString as the most frequent item. For example, in 'mama' we return 'consonant' and {'m': 2, 'a': 2}. You may assume aString is a non-empty string with at least one letter of the alphabet, but you cannot make assumptions about the rest of the contents in the string.

You are not allowed to use the count method or any other Python count libraries such as Counter, mode, etc., or the min() and max() methods. You will not get credit if you used them in your code.

Hints:

- The [str.isalpha\(\)](#) method returns True if all characters in the string are alphabetic and there is at least one character, False otherwise.
- The [str.lower\(\)](#) method returns a copy of the string with all the cased characters converted to lowercase. 'A'.lower() returns 'a'

Preconditions:

| Inputs | | |
|--------|---------|------------------|
| str | aString | Non-empty string |

| Output | |
|-----------|--|
| str, dict | First value is a description of the most frequent letter in numList: 'vowel' or 'consonant' Second value is a dictionary with the frequency count of each item in numList |

Example:

```
>>> frequency('mama')
('consonant', {'m': 2, 'a': 2})
>>> answer = frequency('We ARE Penn State!!!')
>>> answer[0]
'vowel'
>>> answer[1]
{'w': 1, 'e': 4, 'a': 2, 'r': 1, 'p': 1, 'n': 2, 's': 1, 't': 2} # Don't
forget dictionaries are unsorted collections
```

Section 2: Function details

successors(file)

Returns a dictionary whose keys are included words (as is), and values are lists of unique successors to those words. The first word of the *txt* file is always a successor to ".". Be careful with punctuation. You can assume the text in the *txt* file does not contain any contraction words (isn't, don't, I'm, etc) but you cannot make assumptions about the spacing or the presence of non-alphanumerical characters. The starter code already contains the code to read the *txt* file, just make sure the file is in the same directory as your .py file. You must write your code after the file has been read to process the contents.

```
# Open the file and read the contents, the with statement ensures
# the file properly closed after the file operation finishes
with open(file) as f:
    contents = f.read() # reads the entire file, saving data in contents as string
```

Example

Contents of items.txt:

```
We came to learn,eat some pizza and to have fun.
Maybe to learn how to make pizza too!
```

```
>>> print(contents)
'We came to learn,eat some pizza and to have fun.\nMaybe to learn how to make
pizza too!'
```

Hints:

- The [str.isalnum\(\)](#) method returns True if all characters in a string are alphanumeric.
- When $x = \text{'We'}$, the type conversion `list(x)` produces the list `['W', 'e']`

Preconditions:

| Inputs | | |
|--------|------|--|
| str | file | A string that contains the name of a <i>txt</i> file that contains multiple strings. |

| Output | |
|--------|--|
| dict | Keys are all included words and punctuation marks, values all successors of that key |

Example:

Contents of items.txt:

We came to learn,eat some pizza and to have fun.

Maybe to learn how to make pizza too!

```
>>> out=successors('items.txt')
>>> out.keys()
dict_keys(['.', 'We', 'came', 'to', 'learn', ',', 'eat', 'some', 'pizza', 'and',
'have', 'fun', 'Maybe', 'how', 'make', 'too'])
>>> out['.']
['We', 'Maybe']
>>> out['to']
['learn', 'have', 'make']
>>> out['fun']
['.']
>>> out[',']
['eat']
>>> out
# Don't forget dictionaries are unsorted collections
{'.': ['We', 'Maybe'], 'We': ['came'], 'came': ['to'], 'to': ['learn', 'have',
'make'], 'learn': [',', 'how'], ',': ['eat'], 'eat': ['some'], 'some': ['pizza'],
'pizza': ['and', 'too'], 'and': ['to'], 'have': ['fun'], 'fun': ['.'], 'Maybe':
['to'], 'how': ['to'], 'make': ['pizza'], 'too': ['!']}
```

getPosition(num, digit)

Returns an integer that represents the position of the first occurrence of *digit* in *num*. Returns False if *digit* does not appear in *num*. We will define digit positions beginning at 1 and count from the right-most digit of the number. For example, in 264387, the digit 7 is at position 1 and digit 6 is at position 5. You are not allowed to type convert *num* to a string, `str(num)` or to add the digits into a list to traverse or process the number.

Hint:

- Using floor division (`//`) and modulo (`%`) could be helpful here. Floor division by 10 removes the rightmost digit ($456//10 = 45$), while modulo 10 returns the rightmost digit ($456\%10 = 6$)

Preconditions:

| Inputs | | |
|--------|--------------|--|
| int | <i>num</i> | A positive integer |
| int | <i>digit</i> | A positive integer between 0 and 9 (inclusive) |

| Output | |
|--------|---|
| int | Position of <i>digit</i> in <i>num</i> |
| bool | False if <i>digit</i> does not appear in <i>num</i> |

Example:

```
>>> getPosition(1495, 5)
1
>>> getPosition(1495, 1)
4
>>> getPosition(1495423, 4)
3
>>> getPosition(1495, 7)
False
```

Section 2: Function details

hailstone(num)

Returns the hailstone sequence starting at the given number until termination when *num* is 1.

The hailstone sequence is generated as follows:

- If *num* is even, then the next number in the sequence is $num/2$.
- If *num* is odd, then the next number in the sequence is $3 * num + 1$.
- Continue this process until $num = 1$

Preconditions:

| Inputs | | |
|--------|------------|--|
| int | <i>num</i> | A positive integer used to start the hailstone sequence. |

| Output | |
|--------|--|
| list | Hailstone sequence starting at <i>num</i> and ending at 1. |

Examples:

```
>>> hailstone(5)
[5, 16, 8, 4, 2, 1]
>>> hailstone(6)
[6, 3, 10, 5, 16, 8, 4, 2, 1]
```

Section 2: Function details

largeFactor(num)

Returns the largest positive integer that is smaller than *num* and evenly divides *num*.

Preconditions:

| Inputs | | |
|--------|-----|-----------------------------------|
| int | num | A positive integer greater than 1 |

| Output | |
|--------|---|
| int | Largest factor of <i>num</i> that is less than <i>num</i> |

Examples:

```
>>> largeFactor(15) # factors are 1, 3, 5
5
>>> largeFactor(80) # factors are 1, 2, 4, 5, 8, 10, 16, 20, 40
40
>>> largeFactor(13) # factor is 1 since 13 is prime
1
```