Assignment 3: due 8am on Mon, Oct 19, 2020

Summary of Instructions

Note	Read the instructions carefully and follow them exactly
Assignment Weight	7% of your course grade
Due Date and time	8am on Monday, Oct 19, 2020
Important	As outlined in the syllabus, late submissions will not be accepted
	Any files with syntax errors will automatically be excluded from grading. Be sure to test your code before you submit it
	For all functions, both in Part 1 and 2, make sure you've written good docstrings that include type contract, function description and the preconditions if any.

This is an individual assignment. Please review the Plagiarism and Academic Integrity policy presented in the first class, i.e. read in detail pages 15 – 18 of course outline. You can find that file on Brightspace under Course Info. While at it, also review Course Policies on pages 13 and 14.

The goal of this assignment is to learn and practice the concepts covered thus far, in particular: strings (including indexing, slicing and string methods), control structures (if statements and for-loops), use of range function, function design and function calls.

The only collection you can use are strings. You may not use any other collection (such as a list/array, tuple, set or dictionary) in this assignment. Using any of these in a solution to a question constitues changing that question. Consequently, that question will not be graded.

You can make multiple submissions, but only the last submission before the deadline will be graded. What needs to be submitted is explained next.

The assignment has two parts. Each part explains what needs to be submitted. Put all those required documents into a folder called `a3_xxxxxx` where you changed `xxxxxx` to your student number, zip that folder and submit it as explained in Lab 1. In particular, the folder should have the following files:
Part 1: `a3_part1_xxxxxx.py`, `a3_part1_xxxxxx.txt`
Part 2: `a3_part2_xxxxxx.py` and `a3_part2_xxxxxx.txt` and
`declaration-YOUR-FULL-NAME.txt`

About `declaration-YOUR-FULL-NAME.txt` file:

It needs to be a plane text file and it must contain references to any code you used that you did not write yourself, including any code you got from a friend, internet, social media/forums (including Stack Overflow and discord) or any other source or person. The only exclusion from that rule is the following: all the material on the Brightspace of the course, anything from your textbook, anything from python.org and python's help pages. So here is what needs to be written in that file. In every question where you used code from somebody else, you must write:
1. question number
2. copy-pasted parts of the code that were written by somebody else. That includes the code you found/were-given that you then slightly modified.
3. whose code it is: name of a person or place on internet/book where you found it.
While you may not get points for that part of the question, you will not be in position of being accused of plagiarism. Not including `declaration-YOUR-FULL-NAME.txt` will be taken as you declaring that all the code in the assignment was written by you. Any student caught in plagiarism will receive zero for the whole assignment and will be reported to the dean. Finally showing/giving any part of your assignment code to a friend also constitute plagiarism and the same penalties will apply.

If you have nothing to declair, you do not need to submit the file `declaration-YOUR-FULL-NAME.txt`

Both of your programs must run without syntax errors. In particular, when grading your assignment, TAs will first open your file `a3_part1_xxxxxx.py` with IDLE and press Run Module. If pressing Run Module causes any syntax error, the grade for Part 1 becomes zero. The same applies to Part 2, when they open and run file `a3_part2_xxxxxx.py`.

Furthermore, for each of the functions (in Part 1 and Part 2), I have provided one or more tests to test your functions with. For example, you should test function `is_up_monotone` from Part 1 by making a call in Python shell with `is_up_monotone ("12311234","4")`. To obtain a partial mark your function may not necessarily give the correct answer on these tests. But if your function gives any kind of python error when run on the tests provided below, that question will be marked with zero points.

Section 3, contains tests for Part 1. Tests for Part 2 are provided after each question in Part 2. To determine your grade, your functions will be tested both with examples provided in Part 2 and Section 3 and with some other examples. Thus you too should test your functions with more example than what I provided in Part 2 and Section 3.

Use of global variables inside function bodies is not allowed. If you do not know what that means, interpret this to mean that inside of your functions you can only use variables that are created in that function. For example, this is not allowed, since variable `x` is not a parameter of function `a_times(a)` nor is it a variable created in function `a_times(a)`. It is a global variable created outside of all functions.

```
def a_times(a):
   result=x*a
   return result

x=float(input("Give me a number: "))
print(a_times(10))
```

1 Part 1: up-monotone inquiry - 40 points

To clarify Part 1 specifications, I have provided sample tests in Section 3. The behaviour implied by the sample tests should be considered as required specifications in addition to what is explained below.

Description:

A sequence of numbers $x_1, x_2, x_3, x_4....$ is *up-monotone* if $x_1 < x_2 < x_3 < x_4 < .....$

Some positive integers X can be split into smaller pieces, such that the pieces are comprised of consecutive digits of X and such that each piece contains the same number of digits. Given an integer X and given a number of digits d, one can then ask if X can be split into pieces such that each pieces has d digits and such that the resulting sequence of numbers is up-monotone.

Examples:

- For $X = 154152$ and $d = 2$, the answer is *yes*, since $d = 2$ implies the following sequence 15, 41, 52 which is up-monotone.
- For $X = 154152$ and $d = 3$, the answer is *no*, since $d = 2$ implies the following sequence 154, 152 which is not up-monotone.
- For $X = 154152$ and $d = 5$, the answer is *no*, since 154152 cannot be split into five-digit pieces.
- For $X = 137$ and $d = 1$, the answer is *yes*, since since $d = 1$ implies the following sequence 1, 3, 7 which is up-monotone.
- For $X = 137$ and $d = 2$, the answer is *no*, since 137 cannot be split into two-digit pieces.
- For $X = 113$ and $d = 1$, the answer is *no*, since since $d = 1$ implies the following sequence 1, 1, 3 which is not up-monotone.
- For $X = 113$ and $d = 3$, the answer is *yes*, since $d = 3$ implies the following sequence 113 which is up-monotone.

Design, implement and test a Python program which checks to see if a user-supplied positive integer X and a user-specified split d, are such that X can be split into pieces with d digits such that the resulting sequence is up-monotone. Here are the steps that your program should have: (Note that since no collection other than strings is allowed you will be working with X and d as strings)

0. The program greets the user.

1. The program asks the user if they want to test if a number admits an up-monotone split of given size.

2. The program prompts the user to enter a positive integer. If the user enters an invalid input (anything other than a positive integer), the program will repeat the questions starting with step 1.

3. The program prompts the user to enter the number of digits in each piece. The program will verify that the input is valid (a positive integer which is a proper divisor of the number of digits in the first input); If the input is invalid, the program will repeat the questions starting with step 1.

4. Once the valid input is obtained, the program splits the number into pieces of specified length and displays/prints those pieces in one line (separated by commas).

5. Finally, the program reports whether or not the obtained sequence of numbers is up-monotone. If there is only one piece, it is defined to be up-monotone.

For this part, I provided you with starter code in file called `a3_part1_xxxxxx.py`. Begin by replacing `xxxxxx` in the file name with your student number. Then open the file. Your solution (code) for this part must go into that file in the

clearly indicated spaces only. You are not allowed to delete or comment-out or change any parts of the provided code except for the keyword `pass`.

For this part you need to submit two files: `a3_part1_xxxxxx.py` and `a3_part1_xxxxxx.txt`

`a3_part1_xxxxxx.py` needs to contain your program for Part 1 as explained above and `a3_part1_xxxxxx.txt` needs to contain the proof that you tested your two core function from this part, namely `is_up_monotone`.

1.1 The Core Function

Your solution in `a3_part1_xxxxxx.py` must have a function called: `is_up_monotone`. You should design and test function `is_up_monotone` before moving onto designing and coding the `main` part of the program. Here are specifications for that function:

`is_up_monotone` This function has two parameters, a string `X` and a string `d`. These two strings can be assume to be such that each looks like a positive integer and such that the number of digits in `x` is divisible by the integer represented by `d`.

Since `X` and `d` are assumed to be valid i.e. meet the preconditions, they define a sequence of numbers (as specified in the examples above). The function should print that sequence of numbers in such a way that the consecutive numbers are separated by commas as shown in the test cases. Note that there cannot be a comma after the last number in the sequence.

The function should then return True if the sequences is up-monotone and False otherwise. (Reporting, i.e. printing, whether or not the obtained sequence of numbers is up-monotone has to be done in the `main` and not in the body of the `is_up_monotone` function).

Here are two approaches that may help you design this function:

You may find it easier to process the number `X`, which is given as a string, when you split it into pieces.

One approach:

```
start with an empty string to hold the substring, i.e. the split
as you loop through the whole number one digit at a time
    append digits to the substring
    if the substring reaches the desired length
        do something with the substring
        reset the substring to empty to collect the next substring
        reset the substring length count to zero
```

Another approach:

```
figure out how many substrings you want
loop that number of times
    use slices to create the substrings
```

Other suggestions:
-You can use the `isdigit()` string method to determine if a string contains only digits. Type `help(str.isdigit)` in the Python shell for more information.
- You can use the `len()` function to determine the length of a string i.e. the number of digits in `X`.
- If you find getting the commas right in the output tricky (because only commas between numbers are allowed), leave that until everything else is working correctly.

1.2 The User Interaction i.e. the `main` part of the program

Now that you have the function that performs the core functionality, you want to code the communication with the user in the `main`. It is also in the main that you should print the message about whether the sequence is up-monotone or not.

Advice: Leave error checking of the input for last. Begin by building your function first, and then the `main` assuming perfect input. Make sure though to leave enough time at the end to get the input checking details right.

The specifications on how your program needs to behave when communicating with the user should be inferred from the test examples. You will notice that the program is required to display greetings surrounded with stars. One of your functions from Assignment 1, may be helpful for that. You can copy paste that function from your Assignment 1 solution (or mine) to your solution in `a3_part1_xxxxxx.py`.

2 Part 2: A Library of Functions

For this part of the assignment, you are required to write and test several functions (as you did in Assignment 1). You need to save all functions in `a3_part2_xxxxxx.py` where you replace xxxxxx by your student number. You need to test your functions (like you did in Assignment 1) and copy/paste your tests in `a3_part2_xxxxxx.txt`. Thus, for this part you need to submit two files: `a3_part2_xxxxxx.py` and `a3_part2_xxxxxx.txt`

2.1 `sum_odd_divisors(n)` - 5 points

Write a function called `sum_odd_divisors` that takes a integer `n` as input. If `n` is zero, `sum_odd_divisors` returns `None`. Else, `sum_odd_divisors` returns the sum of all the postive odd divisors of n.

```
>>> sum_odd_divisors(-9)
13
>>> sum_odd_divisors(1)
1
>>> sum_odd_divisors(2)
1
>>> sum_odd_divisors(3)
4
>>> sum_odd_divisors(7)
8
>>> sum_odd_divisors(-2001)
2880
```

2.2 `series_sum()` - 5 points

Write a function called `series_sum()` that prompts the user for an non-negative integer n. If the user enters a negative integer the function should return `None` otherwise the function should return the sum of the following series $1000 + 1/1^2 + 1/2^2 + 1/3^2 + 1/4^2 + ... + 1/n^2$ for the given integer n. For example, for $n = 0$, the function should return 1000, for $n = 1$, the function should return 1001, for $n = 2$, the function should return 1001.25, for $n = 3$, the function should return 1001.3611111111111, etc.

```
>>> series_sum()
Please enter a non-negative integer: -10
>>> series_sum()
Please enter a non-negative integer: 0
1000
>>> series_sum()
Please enter a non-negative integer: 5
1001.463611111111
```

2.3 `pell(n)` - 5 points

Pell numbers are a mathematical sequence of numbers that help approximate the value of $\sqrt{2}$ (check out the Wikipedia page on Pell Numbers:
https://en.wikipedia.org/wiki/Pell_number). The sequence is defined recursively as shown in eq. 1.

$$P_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ 2P_{n-1} + P_{n-2} & \text{if } n > 1 \end{cases} \quad (1)$$

Write a function called `pell` that takes one integer parameter `n`, of type `int`. If `n` is negative, `pell` returns `None`. Else, `pell` returns the nth Pell number (i.e. P_n).

```
>>> pell(0)
0
>>> pell(1)
1
>>> pell(2)
2
>>> pell(3)
5
>>> pell(-45)
>>> pell(6)
70
>>> pell(1743)
5326900078425743495271120093406485842060394205546522356059179683018785602043913589772889707680889489315129792427576755779
```

2.4 countMembers(s) - 5 points

In this question you are not allowed to use any of the Python's string methods (i.e. those string functions that you call with dot operator).

Say a character is *extraordinary* if it is one of the following: the lower case letter between `e` and `j` (inclusive), the upper case letters between `F` and `X` (inclusive), numerals between `2` and `6` (inclusive), and the exclamation point (`!`), comma (`,`), and backslash (\)

You are required to count how many times these characters appear in a string.

Write a function called `countMembers` that takes a single input parameter `s`, of type `str`. `countMembers` then returns the number of characters in `s`, that are extraordinary. Therefore, if there are two `X`s in `s`, `countMembers` must count two extraordinary characters (one for each occurrence of `X` in `s`).

```
>>> countMembers("\\")
1
>>> countMembers("2\'")
1
>>> countMembers("1\'")
0
>>> countMembers("2aAb3?eE'_13")
4
>>> countMembers("one, Two")
3
```

2.5 casual_number(s) - 5 points

Imagine a customer in a bank is asked to enter a whole number representing their approximate monetary worth. In that case, some people use commas to separate thousands, millions etc, and some don't. In order to perform a some calculations the bank needs that number as na integer. Write a function, called `casual_number(s)` that has one parameter, `s`, as input where `s` is a string. Function `casual_number(s)` should return an integer representing a number in `s`. If `s` does not look like a number the function should return `None`. Note that if a string in `s` looks like a number but with commas, you may assume that commas are in meaningful places i.e. you may assume that `s` will not be a string like '$1,1,345$'.

```
>>> casual_number("251")
251
>>> casual_number("1,aba,251")
>>> casual_number("1,251")
1251
>>>casual_number("1251")
1251
>>> casual_number("-97,251")
-97251
>>> casual_number("-97251")
-97251
>>> casual_number("-,,,,")
>>> casual_number("--97,251")
>>> casual_number("-")
>>> casual_number("-1,000,001")
-1000001
>>> casual_number("999,999,999,876")
```

```
999999999876
>>> casual_number("-2")
-2
```

2.6 alienNumbers(s) - 5 points

Strange communication has been intercepted between two alien species at war. NASAs top linguists have determined that these aliens use a weird numbering system. They have symbols for various numeric values, and they just add all the values for the symbols in a given numeral to calculate the number. NASA's linguists have given you the following table of symbols and their numeric values. Since they have a lot of these numbers to compute, they want you to write a function that they can use to automate this task.

Symbol	Value
T	1024
y	598
!	121
a	42
N	6
U	1

Thus `a!ya!U!NaU` represents a value of `1095` (see table below for an explanation).

Numeral	Value	Occurrences	Total Value	Running Total
T	1024	0	$0 \times 1024 = 0$	0
y	598	1	$1 \times 598 = 598$	598
!	121	3	$3 \times 121 = 363$	961
a	42	3	$3 \times 42 = 126$	1087
N	6	1	$1 \times 6 = 6$	1093
U	1	2	$2 \times 1 = 2$	1095

Write a function called `alienNumbers` that takes one string parameter `s`, and returns the integer value represented by `s`. Since aliens only know these characters you may assume that no character in `s` outside of this set: `{T,y, !,a, N, U}`. *Callenge:* try to make the whole body of this function only one line long.

```
>>> alienNumbers("a!ya!U!NaU")
1095
>>> alienNumbers("aaaUUU")
129
>>> alienNumbers("")
0
```

2.7 alienNumbersAgain(s) - 5 points

NASA is very pleased with your proof-of-concept solution in the previous question. Now, they've designed a small chip that runs a very restricted version of python - it doesn't have any of the string methods that you are used to. They want you to now rewrite your `alienNumbers` function to run without using any of those string methods you may have otherwise used. Basically, you can use a loop of some sort and any branching you'd like, but none of the string methods. Use *accumulator variable* as we have seen in class.'

Write a function called `alienNumbersAgain`, that takes a single string parameter `s`, and returns the numeric value of the number that `s` represents in the alien numbering system.

```
>>> alienNumbersAgain("a!ya!U!NaU")
1095
>>> alienNumbersAgain("aaaUUU")
129
>>> alienNumbersAgain("")
0
```

The last three questions may be a more challenging to solve especially the last one.

2.8 encrypt(s) - 5 points

Hint: Think of how slicing and reversing of strings can help with this problem. (We saw how to do slicing and reversing at the beginning of Lecture 8)

You want to send a note to your friend in class, and because you want to be respectful in class, you don't want to whip out your phone and send a text or an email (or any other digital communication). Instead, you choose to go old school and write it out on a piece of paper and pass it along to your friend. The problem is, you don't want anyone else to read the note as they pass it along. Luckily, your friend and you have come up with an encryption system so that nobody else can understand your message. Here's how it works: you write out your message backwards (so, `Hello, world` becomes `dlrow ,olleH`). But you don't stop there, because that's too easy to crack - anyone can figure that out! Now that you've written in backwards, Then you start on either side of the string and bring the characters together. So the first and the last characters become the first and the second character in the encrypted string, and the second and the second last characters become the third and the fourth characters in the string, and so on. Thus, `Hello, world` ultimately becomes `dHlerlolwo ,` (notice how all punctuation, special characters, spaces, etc are all treated the same) and `0123456789` becomes `9081726354`.

Write a function called `encrypt`, that has one parameter `s` where `s` is a string and `encrypt` returns a string which is the encrypted version of `s`.

Fun fact in cryptography, `s` is called "clear text" and the encrypted version you return is called "cipher text"

```
>>> encrypt("Hello, world")
'dHlerlolwo ,'
>>>> encrypt("1234")
'4132'
>>> encrypt("12345")
'51423'
>>> encrypt("1")
'1'
>>> encrypt("123")
'312'
>>> encrypt("12")
'21'
>>> encrypt("Secret Message")
'eSgeacsrseetM '
>>> encrypt(",'4'r")
"r,''4"
```

2.9 `weaveop(s)` - 5 points

Hint: Depending on how you plan to solve this problem, accumulator variable initialized as an empty string may help in this question.

Write a function called `weaveop`, that takes a single string parameter (`s`) and returns a string. This function considers every pair of consecutive characters in `s`. It returns a string with the letters `o` and `p` inserted between every pair of consecutive characters of `s`, as follows. If the first character in the pair is uppercase, it inserts an uppercase `O`. If however, it is lowercase, it inserts the lowercase `o`. If the second character is uppercase, it inserts an uppercase `P`. If however, it is lowercase, it inserts the lowercase `p`. If at least one of the character is not a letter in the alphabet, it does not insert anything between that pair. Finally, if `s` has one or less characters, the function returns the same string as `s`.

Do `dir(str)` and check out methods `isalpha` (by typing `help(str.isalpha)` in Python shell), and `isupper`

```
>>> weaveop("aa")
'aopa'
>>> weaveop("aB")
'aoPB'
>>> weaveop("ooo")
'oopoopo'
>>> weaveop("ax1")
'aopx1'
>>> weaveop("abcdef22")
'aopbopcopdopeopf22'
>>> weaveop("abcdef22x")
'aopbopcopdopeopf22x'
>>> weaveop("aBCdef22x")
'aoPBOPCOpdopeopf22x'
>>> weaveop("x")
'x'
>>> weaveop("123456")
'123456'
```

2.10 `squarefree(s)` - 5 points

This may be quite a challenging question to solve. Slicing and remembering that you can ask if two strings `(s1==s2)` are the same are key to a short solution to this problem.

A *squarefree* word is a word that does not contain any subword twice in a row. Examples:
ana is squarefree.
borborygmus is not squarefree, since it has subword, bor twice in a row.
abracadabra is squarefree.
repetitive is not squarefree since subword ti is repeated twice in a row.
grammar is not squarefree since subword m is repeated twice in a row.
gaga is not s squarefree since subword ga is repeated twice in a row.
rambunctious is squarefree.
abcab is squarefree.
abacaba is squarefree.
zrtzghtghtghtq is not squarefree since subword ght is repeated twice (in fact three times, but it is enough to find two repetitions to conclude that the word is not squarefree).
aa is not squarefree since subword a is repeated twice.
zatabracabrac is not squarefree since subword abrac is repeated twice in a row.

Write a function, called **`squarefree`**, that has one parameter, **`s`**, where **`s`** is a string. The function returns True if **`s`** is squarefree and False otherwise.

```
>>> squarefree("")
True
>>> squarefree("a")
True
>>> squarefree("zrtzghtghtghtq")
False
>>> squarefree("abcab")
True
>>> squarefree("12341341")
False
>>> squarefree("44")
False
```

3 Testing your code in Part 1

Here is how you should test your two functions from Part 1 in Python shell.

```
>>> is_up_monotone("12311234","2")
12, 31, 12, 34
False
>>> is_up_monotone ("12311234","4")
1231, 1234
True
>>> is_up_monotone("0012311234","2")
00, 12, 31, 12, 34
False
>>> is_up_monotone("0012311234","5")
00123, 11234
True
>>> is_up_monotone("1","1")
1
True
>>> is_up_monotone("1","1")
1
True
>>> is_up_monotone("734","1")
7, 3, 4
False
>>> is_up_monotone("734","3")
734
True
```

Here is what pressing Run on your program (Part 1) and should give:

```
Python 3.8.0 (v3.8.0:fa919fdf25, Oct 4 2020, 10:23:27)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: /Users/vida/Dropbox/courses/python-iti1120-2020/asssignments/assignment3/a3_part_1-solution.py

**************************************
*                                    *
*  __Welcome to up-monotone inquiry__  *
*                                    *
**************************************

What is your name?     Doctor Manhattan

************************************************************
*                                                          *
*  __Doctor Manhattan, welcome to up-monotone inquiry.__   *
*                                                          *
************************************************************

Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size?    sure
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? sure
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size?   yEs
Good choice!
Enter a positive integer: 123156
Input the split. The split has to divide the length of 123156 i.e. 6
4
4 does not divide 6. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? YES
Good choice!
Enter a positive integer:            123156
Input the split. The split has to divide the length of 123156 i.e. 6
         3
123, 156
The sequence is up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 3.4
The input can only contain digits. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer:    -44
The input has to be a positive integer.Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 3331
Input the split. The split has to divide the length of 3331 i.e. 4
2
33, 31
The sequence is not up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: twenty
The input can only contain digits. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yeS
Good choice!
Enter a positive integer: 4321
Input the split. The split has to divide the length of 4321 i.e. 4
                       1
4, 3, 2, 1
The sequence is not up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer:        4321
Input the split. The split has to divide the length of 4321 i.e. 4
4
4321
The sequence is up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size?    yes
```

```
Good choice!
Enter a positive integer: -2.5
The input can only contain digits. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? 0
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? YES
Good choice!
Enter a positive integer:          0
The input has to be a positive integer.Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 77
Input the split. The split has to divide the length of 77 i.e. 2
2
77
The sequence is up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 22
Input the split. The split has to divide the length of 22 i.e. 2
-5
The split can only contain digits. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? 345
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 345
Input the split. The split has to divide the length of 345 i.e. 3
4
4 does not divide 3. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 345
Input the split. The split has to divide the length of 345 i.e. 3
10
10 does not divide 3. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer:        213411800
Input the split. The split has to divide the length of 213411800 i.e. 9
       5
5 does not divide 9. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 213411800
Input the split. The split has to divide the length of 213411800 i.e. 9
3
213, 411, 800
The sequence is up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? yes
Good choice!
Enter a positive integer: 001230
Input the split. The split has to divide the length of 001230 i.e. 6
3
001, 230
The sequence is up-monotone
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? uf
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size? .... NO
Please enter yes or no. Try again.
Doctor Manhattan, would you like to test if a number admits an up-monotone split of given size?     NO

**********************************
*                                *
*  __Good bye Doctor Manhattan!__  *
*                                *
**********************************
```