Intro to Computer Programming – Fall 2022

Syllabus	Brightspace
Schedule + Class Notes	Ed Forum
Assignments	Poll Everywhere

Assignment #7

For this assignment you will be writing a series of programs that should be stored in their own "py" files. The filenames you should use are listed at the end of each part. When you're finished you should submit your program to the Assignment #7 category inside of NYU Classes.

Part 1a

You have been asked to write a username validation program for a small website. The website has specific rules on what constitutes a valid username, including:

All usernames must be between 8 and 15 characters long Usernames can only contain alphabetic (a-z and A-Z) and numeric characters (0-9) - no special characters or spaces are allowed. The first and last characters in a username cannot be a digit Usernames must contain at least one uppercase character Usernames must contain at least one lowercase character Usernames must contain at least one numeric character

Write a program that asks the user to enter in a username and then examines that username to make sure it complies with the rules above. Here's a sample running of the program - note that you want to keep prompting the user until they supply you with a valid username:

```
Enter a username: craig
* Length of username: 5
* All characters are alpha-numeric: True
* First & last characters are not digits: True
* # of uppercase characters in the username: 0
* # of lowercase characters in the username: 5
* # of digits in the username: 0
Username is invalid, please try again
```

```
Enter a username: craig123
* Length of username: 8
* All characters are alpha-numeric: True
* First & last characters are not digits: False
* # of uppercase characters in the username: 0
* # of lowercase characters in the username: 5
* # of digits in the username: 3
Username is invalid, please try again
```

```
Enter a username: craig 123
* Length of username: 9
* All characters are alpha-numeric: False
* First & last characters are not digits: False
* # of uppercase characters in the username: 0
* # of lowercase characters in the username: 5
* # of digits in the username: 3
Username is invalid, please try again
```

```
Enter a username: Craig123
* Length of username: 8
* All characters are alpha-numeric: True
* First & last characters are not digits: False
* # of uppercase characters in the username: 1
* # of lowercase characters in the username: 4
```

```
* # of digits in the username: 3
Username is invalid, please try again
```

```
Enter a username: Craig123!
* Length of username: 9
* All characters are alpha-numeric: False
* First & last characters are not digits: True
* # of uppercase characters in the username: 1
* # of lowercase characters in the username: 4
* # of digits in the username: 3
Username is invalid, please try again
```

```
Enter a username: Craig123T
* Length of username: 9
* All characters are alpha-numeric: True
* First & last characters are not digits: True
* # of uppercase characters in the username: 2
* # of lowercase characters in the username: 4
* # of digits in the username: 3
Username is valid!
```

Hint: you will need to count the # of uppercase, lowercase and digit characters using some kind of loop.

This program should be named as follows: LastNameFirstName_assign7_part1a.py

Part 1b

The company you are working for was very happy with your username validator, and now they want you to write a password manager for their website. Here are the rules for passwords:

Passwords must be at least 8 characters long (but they do not have an upper limit) Passwords cannot contain the user's username (i.e. if the username is "My1stUsername" the password cannot be "abcMy1stUsername" or "My1stUsernameABC" because the username String can be found in the password String) Passwords must be a mixture of uppercase letters (A-Z), lowercase letters (a-z), digits (0-9) and a select number of special characters (#, $, % and &). The password must contain at least one of each of these types of characters in order to be valid.

You can make a copy of Part 1a and place your password validator code directly after your username validator. Here's a sample running of the program. Note that you need to continually ask the user for a password until they supply a good one.

```
Enter a username: Craig123T
* Length of username: 9
* All characters are alpha-numeric: True
* First & last characters are not digits: True
* # of uppercase characters in the username: 2
* # of lowercase characters in the username: 4
* # of digits in the username: 3
Username is valid!
```

```
Enter a password: foo
* Length of password: 3
* Username is part of password False
* # of uppercase characters in the password: 0
* # of lowercase characters in the password: 3
* # of digits in the password: 0
* # of special characters in the password: 0
Password is invalid, please try again
```

```
Enter a password: foo12345
* Length of password: 8
* Username is part of password False
* # of uppercase characters in the password: 0
* # of lowercase characters in the password: 3
* # of digits in the password: 5
* # of special characters in the password: 0
Password is invalid, please try again
```

```
Enter a password: fooCraig123T
* Length of password: 12
* Username is part of password True
* # of uppercase characters in the password: 2
* # of lowercase characters in the password: 7
* # of digits in the password: 3
* # of special characters in the password: 0
Password is invalid, please try again
```

```
Enter a password: Craig123Tfoo
* Length of password: 12
* Username is part of password True
* # of uppercase characters in the password: 2
* # of lowercase characters in the password: 7
* # of digits in the password: 3
* # of special characters in the password: 0
Password is invalid, please try again
```

```
Enter a password: foo123ABC
* Length of password: 9
* Username is part of password False
* # of uppercase characters in the password: 3
* # of lowercase characters in the password: 3
* # of digits in the password: 3
* # of special characters in the password: 0
Password is invalid, please try again
```

```
Enter a password: foo123ABC#
* Length of password: 10
* Username is part of password False
* # of uppercase characters in the password: 3
* # of lowercase characters in the password: 3
* # of digits in the password: 3
* # of special characters in the password: 1
Password is valid!
```

Hint: you will need to count the # of uppercase, lowercase and special characters using some kind of loop. Also, refer to the ASCII table as needed! - you may need to convert portions of your password into their ASCII locations using the ord() function!

This program should be named as follows: LastNameFirstName_assign7_part1b.py

Part 2a

Numerology is the “study of the purported mystical or special relationship between a number and observed or perceived events.” It has been used throughout human history as a way to attach meaning to a name, object or event using mathematics. It is considered a “pseudoscience” by modern scientists since it has no basis in observable phenomena. With that said, it makes a great programming challenge so we’re going to go with it! :)

What you want to do for this project is to ask the user to type in their name. Next, you will need to use a technique called “theosophical reduction” to convert their name into a number. With this technique we assign each letter of the alphabet its own number. For example, the letter “a” is equal to the number 1. “b” = 2, “c” = 3, “z” = 26, etc. You should ignore non-alphabetic characters (i.e. numbers, spaces and special characters)

Once you’ve gotten all of the letters converted into numbers you can add them up into one single number. This is the “numerology number” for the name that the user entered.

So for the name “craig” the numerology number would be:

```
c = 3
r = 18
a = 1
i = 9
g = 7

3 + 18 + 1 + 9 + 7 = 38
```

Here’s are a few sample runnings of this program:

```
Name: craig
Your 'cleaned up' name is: craig
Your 'cleaned up' name reduces to:
3 + 18 + 1 + 9 + 7 = 38
```

```
Name: craigkapp
Your 'cleaned up' name is: craigkapp
Your 'cleaned up' name reduces to:
3 + 18 + 1 + 9 + 7 + 11 + 1 + 16 + 16 = 82
```

```
Name: rumple stil skin
Your 'cleaned up' name is: rumplestilskin
Your 'cleaned up' name reduces to:
18 + 21 + 13 + 16 + 12 + 5 + 19 + 20 + 9 + 12 + 19 + 11 + 9 + 14 = 198
```

```
Name: !rumple!stil!skin
Your 'cleaned up' name is: rumplestilskin
Your 'cleaned up' name reduces to:
18 + 21 + 13 + 16 + 12 + 5 + 19 + 20 + 9 + 12 + 19 + 11 + 9 + 14 = 198
```

```
Name: pikachu!pikachu!
Your 'cleaned up' name is: pikachupikachu
Your 'cleaned up' name reduces to:
16 + 9 + 11 + 1 + 3 + 8 + 21 + 16 + 9 + 11 + 1 + 3 + 8 + 21 = 138
```

```
Name: PIKACHUpikachu
Your 'cleaned up' name is: pikachupikachu
Your 'cleaned up' name reduces to:
16 + 9 + 11 + 1 + 3 + 8 + 21 + 16 + 9 + 11 + 1 + 3 + 8 + 21 = 138
```

Some hints:

- Convert the user’s name to all lowercase before you do anything else
- Remove any spaces, numbers or special characters from the name to ensure that you are only working with the letters A-Z
- The ord() function may be userful to convert each character into an ASCII index

This program should be named as follows: LastNameFirstName_assign7_part2a.py

Part 2b

Classic numerology ascribes meaning to the following numbers:

- 0 = emptiness, nothingness, blank
- 1 = independence, loneliness, creativity, originality, dominance, leadership, impatience
- 2 = quiet, passive, diplomatic, co-operation, comforting, soothing, intuitive, compromising, patient
- 3 = charming, outgoing, self expressive, extroverted, abundance, active, energetic, proud
- 4 = harmony, truth, justice, order discipline, practicality
- 5 = new directions, excitement, change, adventure
- 6 = love, harmony, perfection, marriage, tolerance, public service
- 7 = spirituality, completeness, isolation, introspection
- 8 = organization, business, commerce, new beginnings
- 9 = romantic, rebellious, determined, passionate, compassionate

However, you might recall from the previous problem that the sample input ("craig") reduced to the number 38. 38 is not on the "personality trait" lookup table above, so we need to further reduce the number by adding up its individual digits like so:

```
3 + 8 = 11
```

The number 11 is not on the personality traits table, so we have to further reduce it:

```
1 + 1 = 2
```

The number 2 is on the table, so we can print out to the user what their traits are based on this number.

Note that it might take a few tries to reduce the user's number to a number that is on the personality trait listing. You might want to think about building in a "while" loop that handles this process.

Here are a few sample runnings of this program:

```
Name: craig
Your 'cleaned up' name is: craig
Your 'cleaned up' name reduces to:
3 + 18 + 1 + 9 + 7 = 38
Further reduction: 11
Further reduction: 2
This name means ...Quiet


Name: pikachu
Your 'cleaned up' name is: pikachu
Your 'cleaned up' name reduces to:
16 + 9 + 11 + 1 + 3 + 8 + 21 = 69
Further reduction: 15
Further reduction: 6
This name means ...Love


Name: charmander!
Your 'cleaned up' name is: charmander
Your 'cleaned up' name reduces to:
3 + 8 + 1 + 18 + 13 + 1 + 14 + 4 + 5 + 18 = 85
Further reduction: 13
Further reduction: 4
This name means ...Harmony
```

```
Name: rumplestilskin
Your 'cleaned up' name is: rumplestilskin
Your 'cleaned up' name reduces to:
18 + 21 + 13 + 16 + 12 + 5 + 19 + 20 + 9 + 12 + 19 + 11 + 9 + 14 = 198
Further reduction: 18
Further reduction: 9
This name means ...Romantic
```

Some hints:

- Attempt to reduce the user's name one time before you attempt to further reduce it (i.e. my name reduces to 38 the first time - get your name to reduce like this as well, and don't worry about further reducing the name until you understand how to do it the first time)
- Try to use the ord() function to convert a single character into its ASCII equivalent. This should help in the conversion process.
- Once you have reduced the name you should test to see if it is one of the "special" numbers listed above. If so, tell the user what their traits are and end the program. If the number is not one of the numbers above then you need to further reduce it. Hint: this should be done using a "while" loop to ensure that you reduce the number as far as it can go.

This program should be named as follows: LastNameFirstName_assign7_part2b.py

Part 3a

For this part you will be writing a series of functions that can be used as part of a "cryptography" program. Here are the functions you will be writing as well as some sample code that you use use to test your work.

Function #1

```
# function:    ascii_shift
# input:       a word to shift (String) and an amount to shift by (integer)
# processing: shifts each character in the supplied word to another position in the ASCII
#              table. the new position is dictated by the supplied integer.  for example,
#              if word = "apple" and num=1 the newly generated word would be:
#
#              bqqmf
#
#              because we added +1 to each character. if we were to call the function with
#              word = "bqqmf" and num=-1 the newly generated word would be:
#
#              apple
#
#              because we added -1 to each character, which shifted each character down by
#              one position on the ASCII table.
#
#              in the event that an empty string is supplied no shift will occur and an empty
#              string should be returned
#
# output:      returns the newly generated word
```

Sample Program

```
word = "ABCDEFG"
for i in range(-5, 6):
    print ("ASCII shifting", word, "by", i, "=>", ascii_shift(word, i))
```

Sample Output

```
ASCII shifting ABCDEFG by -5 => <=>?@AB
ASCII shifting ABCDEFG by -4 => =>?@ABC
ASCII shifting ABCDEFG by -3 => >?@ABCD
ASCII shifting ABCDEFG by -2 => ?@ABCDE
```

```
ASCII shifting ABCDEFG by -1 => @ABCDEF
ASCII shifting ABCDEFG by 0 => ABCDEFG
ASCII shifting ABCDEFG by 1 => BCDEFGH
ASCII shifting ABCDEFG by 2 => CDEFGHI
ASCII shifting ABCDEFG by 3 => DEFGHIJ
ASCII shifting ABCDEFG by 4 => EFGHIJK
ASCII shifting ABCDEFG by 5 => FGHIJKL
```

Function #2

```
# function:   shift_right
# input:      a word to shift (String)
# processing: shifts all characters in the string to the right. the last character in the string
#             will be shifted to the beginning of the string.  for example:
#
#             apple -> eappl
#
#             in the event that an empty string is supplied no shift will occur and an empty
#             string should be returned
#
# output:     returns the newly generated word
```

Sample Program

```
word = "hello world!"
print("Shifting right", word, "=>", shift_right(word))
Sample output
Shifting right hello world! => !hello world
```

Function #3

```
# function:   shift_left
# input:      a word to shift (String)
# processing: shifts all characters in the string to the left. the first character in the string
#             will be shifted to the end of the string.  for example:
#
#             apple -> pplea
#
#             in the event that an empty string is supplied no shift will occur and an empty
#             string should be returned
#
# output:     returns the newly generated word
```

Sample Program

```
word = "hello world!"
print("Shifting left", word, "=>", shift_left(word))
```

Sample output

```
Shifting left hello world! => ello world!h
```

Function #4

```
# function:   flip
# input:      a word to flip (String)
# processing: flips the first half of the string with the second half of the string.
#             if the string has an even number of characters this function will work as follows:
#
#             ABCD -> CDAB
#
```

```
#             if the string has an odd number of characters this function will work as follows:
#
#             ABCDE -> DECAB
#
#             in the event that an empty string is supplied no flip will occur and an empty
#             string should be returned
#
# output:     returns the newly generated word
```

Sample Program

```
word = "ABCDEFG"
print ("Flipping", word, "=>", flip(word))

word = "123456"
print ("Flipping", word, "=>", flip(word))
```

Sample Output

```
Flipping ABCDEFG => EFGDABC
Flipping 123456 => 456123
```

Function #5

```
# function:   add_letters
# input:      a word to scramble (String) and a number of letters (integer)
# processing: adds a number of random letters (A-Z; a-z) after each letter
#             in the supplied word. for example, if word="cat" and num=1
#             we could generate any of the following:
#             cZaQtR
#             cwaRts
#             cEaett
#
#             if word="cat" and num=2 we could generate any of the following:
#             cRtaHFtui
#             cnnaNYtjn
#             czAaAitym
#
# output:     returns the newly generated word

def add_letters(word, num):

        # function code goes here!
```

Sample Program

```
# define original word
original = "Hello!"

# loop to demonstrate the function
for num in range(1, 5):

    # scramble the word using 'num' extra characters
    scrambled = add_letters(original, num)

    # output
    print ("Adding", num, "random characters to", original, "->", scrambled)
```

Sample Output

```
Adding 1 random characters to Hello! -> HdeulHlHom!t
Adding 2 random characters to Hello! -> HTLedklFNljioMH!bi
```

```
Adding 3 random characters to Hello! -> HHHZeZrflqSflzOiosNU!jBk
Adding 4 random characters to Hello! -> HFtRKeivFllRNlUlGTaooYwoH!JpXL
```

Hint: you will need to use a loop to generate the required # of random characters, and you will (obviously) need to use some random number functions as well. Think about the algorithm before you start coding! Draw out the steps you think you need to take on a piece of paper. For example: “Start with the first character in the source word. Then generate ‘num’ new random characters and concatenate these characters. Then move onto the next character in the source word and repeat the process”.

Function #6

```
# function:   delete_characters
# input:      a word to analyze (String) and the number of characters to remove (integer)
# processing: the function starts at the first position in the supplied word and keeps it.
#             it then removes "num" characters from the word. the process is repeated again
#             if the word contains additional characters - the next character is kept
#             and "num" more characters are removed.  For example, if word="cZaYtU" and
#             num=1 the function will generate the following:
#
#             cat (keeping character 0, removing character 1, keeping character 2, removing
#                  character 3, keeping character 4, removing character 5)
#
# output:     returns the newly unscrambed word

def delete_characters(word, num):

        # function code goes here!
```

Sample Program

```
word1 = "HdeulHlHom!t"
word2 = "HTLedklFNljioMH!bi"
word3 = "HHHZeZrflqSflzOiosNU!jBk"
word4 = "HFtRKeivFllRNlUlGTaooYwoH!JpXL"

unscrambled1 = delete_characters(word1, 1)
print ("Removing 1 characer from", word1, "->", unscrambled1)

unscrambled2 = delete_characters(word2, 2)
print ("Removing 2 characers from", word2, "->", unscrambled2)

unscrambled3 = delete_characters(word3, 3)
print ("Removing 3 characers from", word3, "->", unscrambled3)

unscrambled4 = delete_characters(word4, 4)
print ("Removing 4 characers from", word4, "->", unscrambled4)
```

Sample Output

```
Removing 1 character from HdeulHlHom!t -> Hello!
Removing 2 character from HTLedklFNljioMH!bi -> Hello!
Removing 3 character from HHHZeZrflqSflzOiosNU!jBk -> Hello!
Removing 4 character from HFtRKeivFllRNlUlGTaooYwoH!JpXL -> Hello!
Hint: String slicing may make your life a lot easier when writing this function!
```

This program should be named as follows: LastNameFirstName_assign7_part3a.py

Part 3b

Now you are going to write an “encoder / decoder” program that makes use of your six cryptographic functions. Begin by writing a program that continually asks the user to enter in an encoding pattern and a word to encode. The user should also be able to end the program using a sentinel value.

If the user chooses to encode a word you should do the following:

Ask the user for an “encoding pattern” string. This string will contain instructions on how to encode / decode a string. The valid commands that this string can contain are the following:

- ‘A’ = add 1 character in between all characters
- ‘X’ = delete 1 character in between all characters
- ‘F’ = flip the string
- ‘U’ = ASCII shift all characters by +1
- ‘D’ = ASCII shift all characters by -1
- ‘L’ = Shift all characters to the left by 1
- ‘R’ = Shift all characters to the right by 1

Next, ask them to enter in a phrase that they want to encode.

Finally, apply the desired encoding pattern to their string, giving them feedback during the process. Apply the pattern one operation at a time, from left to right. Any invalid or unrecognized commands should be ignored.

Hint: use your functions! Hint: you may need an accumulator variable to keep track of the current state of the word

Here are a few sample runnings of the program:

```
Enter an encoding pattern, 'end' to end: AFUUURRR
Enter a word to encode/decode: Hello World!
* Added 1 character: HYeflUlhox FWLoorkladM!V
* Flipped: WLoorkladM!VHYeflUlhox F
* ASCII shifted up: XMppslmbeN"WIZfgmVmipy!G
* ASCII shifted up: YNqqtmncfO#XJ[ghnWnjqz"H
* ASCII shifted up: ZOrrunodgP$YK\hioXokr{#I
* Shifted right: IZOrrunodgP$YK\hioXokr{#
* Shifted right: #IZOrrunodgP$YK\hioXokr{
* Shifted right: {#IZOrrunodgP$YK\hioXokr


Enter an encoding pattern, 'end' to end: LLLDDDFX
Enter a word to encode/decode: {#IZOrrunodgP$YK\hioXokr
* Shifted left: #IZOrrunodgP$YK\hioXokr{
* Shifted left: IZOrrunodgP$YK\hioXokr{#
* Shifted left: ZOrrunodgP$YK\hioXokr{#I
* ASCII shifted down: YNqqtmncfO#XJ[ghnWnjqz"H
* ASCII shifted down: XMppslmbeN"WIZfgmVmipy!G
* ASCII shifted down: WLoorkladM!VHYeflUlhox F
* Flipped: HYeflUlhox FWLoorkladM!V
* Deleted 1 character: Hello World!


Enter an encoding pattern, 'end' to end: end
Enter an encoding pattern, 'end' to end: FDDLLLA
Enter a word to encode/decode: HappyBirthdaySadKitten
* Flipped: aySadKittenHappyBirthd
* ASCII shifted down: `xR`cJhssdmG`ooxAhqsgc
* ASCII shifted down: _wQ_bIgrrclF_nnw@gprfb
* Shifted left: wQ_bIgrrclF_nnw@gprfb_
* Shifted left: Q_bIgrrclF_nnw@gprfb_w
* Shifted left: _bIgrrclF_nnw@gprfb_wQ
* Added 1 character: _ibPIcgxrirtcQlvFs_Gnhnqwi@sgbpmrofnbM_VwEQH


Enter an encoding pattern, 'end' to end: XRRRUUF
Enter a word to encode/decode: _ibPIcgxrirtcQlvFs_Gnhnqwi@sgbpmrofnbM_VwEQH
* Deleted 1 character: _bIgrrclF_nnw@gprfb_wQ
* Shifted right: Q_bIgrrclF_nnw@gprfb_w
* Shifted right: wQ_bIgrrclF_nnw@gprfb_
* Shifted right: _wQ_bIgrrclF_nnw@gprfb
* ASCII shifted up: `xR`cJhssdmG`ooxAhqsgc
```

```
* ASCII shifted up: aySadKittenHappyBirthd
* Flipped: HappyBirthdaySadKitten

Enter an encoding pattern, 'end' to end: end
```

This program should be named as follows: LastNameFirstName_assign7_part3b.py

Part 4

For this part you will be implementing a series of commonly used String functions and methods by writing your own functions. These functions should behave just like their commonly used counterparts - here are some IPO notation blocks & sample code to get you started:

```
# function:   string_length
# input:      a word (String)
# processing: computes the length of the supplied String (without using the len() function)
# output:     returns the length of the string (integer)

# sample code:
print ( string_length("apple") )        # 5
print ( string_length("pear")  )        # 4
print ( string_length("")      )        # 0
```

```
# function:   string_isalpha
# input:      a word (String)
# processing: determines if the supplied String contains all alphabetic characters (A-Z,a-z)
#             DO NOT USE THE "isalpha()" METHOD or any other String methods! You may
#             use the 'ord' and 'chr' functions though.
# output:     returns True of False (boolean)

# sample code:
print ( string_isalpha("apple")     )   # True
print ( string_isalpha("pear!")     )   # False
print ( string_isalpha("123")       )   # False
print ( string_isalpha("123 AbC")   )   # False
print ( string_isalpha("abc1")      )   # False
print ( string_isalpha("")          )   # False
```

```
# function:   string_adjustcase
# input:      a word (String) and a case type (String)
# processing: consults the case type (either "upper" or "lower") and converts all alphabetic
#             characters in the word to their desired equivalents.  will return the original
#             phrase if the case type is invalid.
#             DO NOT USE THE "lower()" METHOD OR "str.lower()"! or any
#             other String methods! you may use the 'ord' and 'chr' functions though, if you wish.
# output:     returns a new copy of the String

# sample code:
# sample code:
print ( string_adjustcase("apple", "upper")     )               # APPLE
print ( string_adjustcase("APPLE", "lower")     )               # apple
print ( string_adjustcase("aPPlE", "lower")     )               # apple
print ( string_adjustcase("APPLE", "pikachu")     )             # APPLE
print ( string_adjustcase("Hello World!", "lower")     )        # hello world!
print ( string_adjustcase("", "lower")     )                    # nothing prints
```

```
# function:   string_capitalize
# input:      a phrase (String)
# processing: capitalizes the first alphabetic character of each word in a string,
#             and lowercases all other alphabetic characters.  For example:
```

```
#
#             "hello world" -> "Hello World"
#             "HELLO WORLD" -> "Hello World"
#
#             DO NOT USE THE "capitalize()" or "title()" METHODS,
#             or any other String methods!
#             You can use 'ord', 'chr' or any other functions that you wrote for this
#             part of the assignment.
#
#             Hint: you might need to iterate over the supplied phrase and test to see
#             if a character is preceded by a space in order to solve this question.
# output:     returns a new copy of the String

# sample code:
print( string_capitalize("happy birthday sad kitten") )
# Happy Birthday Sad Kitten

print( string_capitalize("every word in this phrase should start with a capital letter") )
# Every Word In This Phrase Should Start With A Capital Letter

print( string_capitalize("EVERYTHING IS UPPERCASE ALREADY!") )
# Everything Is Uppercase Already!

print( string_capitalize("cRaZy MIxeD cAse") )
# Crazy Mixed Case
```