Introduction

Many websites and apps require you to create a password to register for, and log in to, a personal account. Most of these platforms have restrictions on the password you create so that it is not easy for hackers to crack. These restrictions are usually based on the number of characters in the password and the different groups of character types used in the password. Short passwords, such as hello123, are easy to crack. Passwords with only one type of character (i.e. only lowercase letters or only digits), such as 123456789, are also easy to crack. This is why most websites and apps will require that you create a password that is long and contains a variety of character types. They typically show you the "strength" of the password you type in so that you know how safe your password is from being hacked.

For this assignment, you will be creating a Python program to calculate the strength of a password based on its length and number of groups of character types used in the password. You must follow our instructions carefully to calculate the strength of a given password. The way you implement the code is up to you but it must follow the system explained below in the instructions to ensure your code works as expected. Note that each platform has a slightly different way of calculating password strength and the system used for this assignment is not necessarily the same as that from anywhere else.

In this assignment, you will get practice with:

- Creating functions
- Calling functions and using return values
- Conditionals and loops
- User input in a loop
- Implementing code based on given instructions
- Importing files to call functions from other files

Files

For this assignment, you must create three (3) Python files named: *password.py*, *generate.py*, and *login.py*. These three files must be saved in the same folder to work correctly since login.py will need to access the other two files. **Make sure you name these files exactly as instructed including having them all lowercase**.

Some example inputs and the expected outputs are provided later in these instructions. You are also encouraged to test your program with many other possible inputs to ensure that your code works in many different cases. Avoid testing with any of your actual account passwords in case anybody sees them!

1) password.py

In the password.py file, you need to create two functions. One will calculate the strength of a given password and the other one will be a helper function to count the number of character groups used by the password. We will start with the helper function and then call it from the strength calculator function.

First, define a function count_groups(*pwd*) which takes in a single parameter *pwd* which is a String representing a password. The function must count and **return** (this does not mean print it out) the number of character groups used in *pwd*. The four character groups are depicted in Figure 1 below. Note that not all characters are included in one of these four groups. For example, brackets/parentheses, underscores, and special characters (i.e. é and ö) are not included in any group, but they are still allowed to be used in passwords. Thus, it is possible that a password could yield 0 groups if it consists only of characters that do not belong to any of the four groups.

For example, count_groups("abc123") should return 2 because the password "abc123" contains characters from 2 groups: lowercase letters and digits.

Group Name / Description	Characters in Group
Lowercase letters	abcdefghijklmnopqrstuvwxyz
Uppercase letters	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Digits	0123456789
Symbols	!@#$%^&*/?-+=,.|~

Figure 1. The groups of character types used in calculating a password's strength.

The second function in this file must be password_strength(*pwd*) which takes in a single parameter *pwd* which is a String representing a password. This function must calculate the strength of the password *pwd* and **return** the strength value (an integer from 0 to 5 inclusive). The password strength must be calculated by examining the length of the password and the number of character groups included in the password; using the helper function to get the number of groups.

Passwords are not allowed to contain spaces, tab characters (\t), or newline characters (\n) and must be given a strength of 0 if they contain any of those invalid characters, regardless of the password's other characters. You must follow the rules shown in Figure 2 below to compute the strength. Some examples are provided in Figure 3.

Password Length	Num Groups in Password	Password Strength
< 5	0 – 4	0
5 – 8	0 – 1	1
	2 – 3	2
	4	3
9 – 12	0 – 1	2
	2 – 3	3
	4	4
> 12	0 – 1	3
	2 – 3	4
	4	5
any	any; but contains a space, tab, or newline character	0

Figure 2. The system to calculate a password's strength based on its length and the number of character groups used in the password.

Password	Strength	Reason
Hello\tWorld!	0	Tab character (\t)
PYTHON	1	5-8 chars; 1 group (uppercase letters)
opensesame	2	9-12 chars; 1 group (lowercase letters)
Pythonw1z	3	9-12 chars; 3 groups (lowercase letters, uppercase letters, digits)
programming1026	4	>12 chars; 2 groups (lowercase letters and digits)
W3lcome2W3st3rn!	5	>12 chars; 4 groups (lowercase letters, uppercase letters, digits, symbols)

Figure 3. Some examples of passwords and their calculated strengths along with a brief explanation of the reason for the assigned strength.

2) generate.py

In generate.py, you must create one function called generate_password(*length*). This function must randomly generate a password of length *length* composed of randomly selected characters from the four character groups (see below for information on how to do this). The function must build the password one character at a time and then return the complete password.

You will need to import the random library into this file by including the following line at the top of the file:
import random

The random library is a built-in library that contains many functions for generating random numbers and making random selections. We will be using a function (random.choice) from this library to make a random selection from a string consisting of all characters from the four groups combined. So, you will need to create a large string, *ALL_CHARS*, with all those characters (all lowercase letters, uppercase letters, digits, and the symbols specified in Figure 1). Then you will need to use random.choice(*ALL_CHARS*). This "choice" function in the random library will randomly select a character from the given string and return it. Do this repeatedly and concatenate each one to a string (initially empty) so that it gets built up with *length* randomly selected characters. Return this string once the loop finishes. For more information on the random.choice() function, see the official doc here: https://docs.python.org/3/library/random.html#random.choice

3) login.py

The third file you need to create is login.py. This file will be used to prompt the user to create a new password or to use the password generator to randomly generate a password. This file must import both of the previous files you created so that the functions in those files can be called from this file. To link password.py and generate.py to your login.py file, you must save all 3 files in the same folder and include the following lines of code at the top of login.py:

from password import *
from generate import *

Note: you should **not** include any import lines in password.py or generate.py (except for importing the "random" library in generate.py).

In login.py, implement a function called process_password(*min_strength*). In this function, you must prompt the user for input within a loop that continues until the user creates or randomly generates a password which has a strength of *min_strength* or higher.

When the user types input, check if they typed a single X or x (either case) and if so, generate a random password of length 15 using the generate_password function you created in the generate.py file. If they did not enter X or x, then take their inputted text as a possible password. Call the password_strength function you created in the password.py file to check the strength of the user's new password. If this strength is greater than or equal to *min_strength*, end the loop; otherwise the loop must continue running to prompt the user for input again.

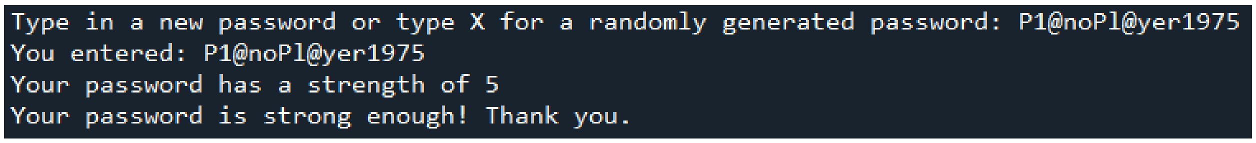
This file must display messages based on the user inputted password:

- When the user is prompted for input, display a message that they must type in a new password or type an X for a randomly generated password.

- If they typed in X or x (either case), display the randomly generated password along with the strength of that password.
- If they did not type in X or x, display the password they just typed in along with the strength of that password.
- If the strength is less than *min_strength*, show a message that the password is "not strong enough" (ensure these words are used exactly like this) and prompt the user to type in a new password again.
- If the strength is greater than or equal to *min_strength*, show a message that the password is strong enough and let the program terminate.

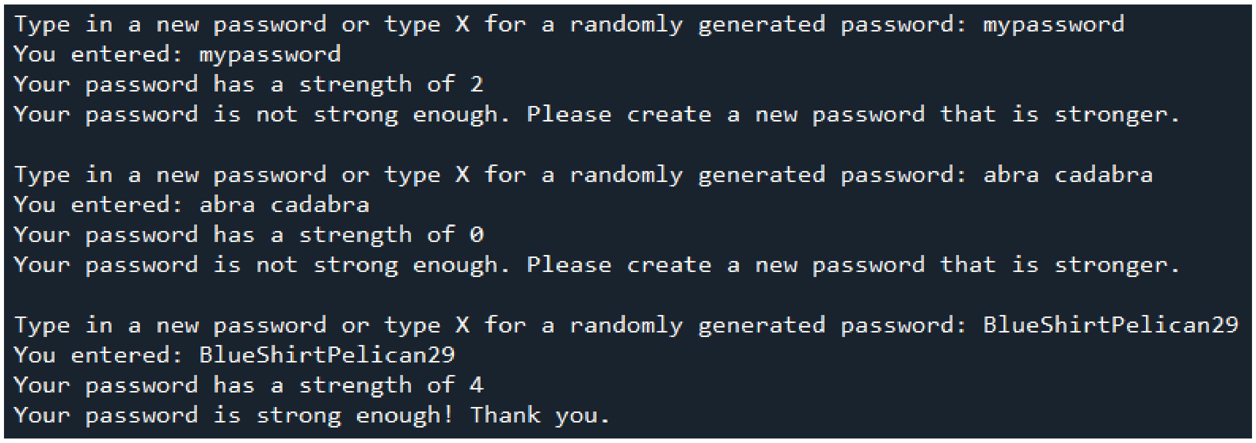
Test Case Examples

Below are three examples of login.py input and output when running process_password(3) (i.e. the *min_strength* parameter is 3 for these examples).



```
Type in a new password or type X for a randomly generated password: P1@noPl@yer1975
You entered: P1@noPl@yer1975
Your password has a strength of 5
Your password is strong enough! Thank you.
```

Example 1: the first password typed in has a strength greater than 3 so the program terminates immediately without asking for a new password.

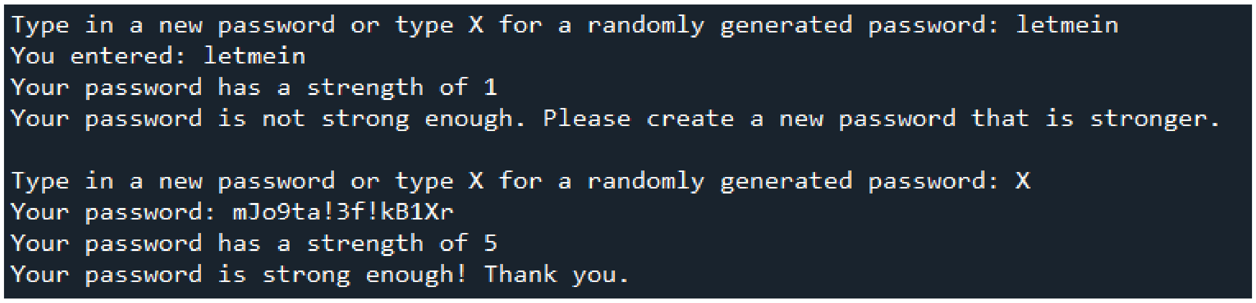


```
Type in a new password or type X for a randomly generated password: mypassword
You entered: mypassword
Your password has a strength of 2
Your password is not strong enough. Please create a new password that is stronger.

Type in a new password or type X for a randomly generated password: abra cadabra
You entered: abra cadabra
Your password has a strength of 0
Your password is not strong enough. Please create a new password that is stronger.

Type in a new password or type X for a randomly generated password: BlueShirtPelican29
You entered: BlueShirtPelican29
Your password has a strength of 4
Your password is strong enough! Thank you.
```

Example 2: the user's first password is too weak so they are prompted to make a new one. The second password contains a space making it have a strength of 0 so they are prompted again to make a new one. The third attempt is successful as the password's strength is greater than 3 so the program then terminates.



```
Type in a new password or type X for a randomly generated password: letmein
You entered: letmein
Your password has a strength of 1
Your password is not strong enough. Please create a new password that is stronger.

Type in a new password or type X for a randomly generated password: X
Your password: mJo9ta!3f!kB1Xr
Your password has a strength of 5
Your password is strong enough! Thank you.
```

Example 3: the user's first password is too weak so they are prompted to make a new one. The user then types X to indicate that they want a randomly generated password. The random password is successful as the password's strength is greater than 3 (these randomly generated passwords will ***always*** *yield a strength of at least 3 since they are 15 characters long; and they will* ***usually*** *have a strength of 5 given the random character selection) so the program terminates.*

Do not have function calls or any code outside of functions. Writing your code in global space, outside of functions, will likely cause the autograder to fail and you will not receive marks for those tests. Make sure your code is contained within functions and that you do not leave any function calls at the bottom of your files.

Provided File

You are given a short file called driver.py which you can use to test that your files are working correctly. Make sure you save driver.py in the same folder as the other files. If you have any code in global space in your files, it will cause driver.py to not run properly so remove or comment out any code in global space in your files (except for import lines and global variables declared at the top).

Tips and Guidelines

- Constants must be named with all uppercase letters, i.e. *DIGITS*.
- Variables should be named in all lowercase and contain underscores between words for multi-word names, i.e. *group* or *num_groups*.
- Add comments throughout your code to explain what each section of the code is doing and/or how it works.
- You are allowed to create additional (helper) functions if you wish. They are not required but they are allowed.
- When you submit your code on Gradescope, look at the autograded test scores. If any tests are failing (i.e. 5/7 means that out of 7 tests, 5 are passing and 2 are failing), it means that your code isn't completely correct.
- Test your code with a variety of different input values and check that it's functioning correctly for all cases. It is your responsibility to test your code for correctness.

Rules

- Read and follow the instructions carefully.
- Submit all the Python files described in the Files section of this document and no additional files.
- Submit the assignment on time. Late submissions will not be accepted or will be marked as 0, except where late coupons are used.
- Forgetting to submit a finished assignment is **not** a valid excuse for missing a due date.
- Make regular backups of your work and store it in multiple places, i.e. USB key, on the cloud, etc. Experiencing technical issues resulting in lost or corrupted files is **not** a valid excuse for missing a due date.
- Submissions must be done on Gradescope. They will **NOT** (under any circumstances) be accepted by email.

- You may re-submit your code as many times as you would like. Gradescope uses your last submission as the one for grading by default. There are no penalties for re-submitting. However, re-submissions that come in after the due date **will** be considered late and marked as 0, unless you have enough late coupons remaining, in which case late coupons will be applied.
- The top of **each** file you create must include your name, your Western ID, the course name, the date on which you **created** the file (no need to change it each time you modify your code), as well as a description of what the file does. This information is **required**. You must follow this format:

```
"""
******************************
CS 1026 - Assignment X – Interest Calculator
Code by: Jane Doe
Student ID: jdoe123
File created: October 12, 2024
******************************
This file is used to calculate monthly principal and interest amounts for a
given mortgage total. It must calculate these values over X years using
projected variations in interest rates. The final function prints out all the
results in a structured table.
"""
```

- Assignments will be run through a similarity checking software to check for code that looks very similar to that of other students. Sharing or copying code in any way is considered plagiarism and may result in a mark of zero (0) on the assignment and/or reported to the Dean's Office. Plagiarism is a serious offence. Work is to be done **individually**.

What constitutes plagiarism

Plagiarism, or academic dishonesty, comes in a variety of forms including:

- sending any portion of your code to peer(s)
- letting peer(s) look at your code

- using any portion of code from peer(s)
- using ChatGPT or other AI platforms to generate any portion of code or comments
- using code that you found online
- paying someone to do your work
- being paid to do someone else's work
- leaving your work on a computer unattended where peers may be able to find it
- uploading your code to an online repository where peers may be able to access it

Submission

Due: Wednesday, October 23, 2024 at 11:59pm

You must submit the three (3) files (*password.py*, *generate.py*, and *login.py*) to the Assignment 2 submission page on Gradescope. There are several tests that will automatically run when you upload your files. You will see the score of the tests but you will not see what is being tested. It is recommended that you create your own test cases to check that the code is working properly in many different scenarios.

Marking Guidelines

The assignment will be marked as a combination of your auto-graded tests and manual grading of your comments, formatting, etc. Below is a breakdown of the marks for this assignment:

[17 marks] Auto-graded tests

[1 mark] Header section with name, student ID, course info, creation date, and description of file

[1 mark] Comments throughout code

[1 mark] Meaningful variable names

Total: 20 marks

IMPORTANT:

A mark of zero (0) will be given for tests that do not run on Gradescope. It is your responsibility to ensure that your code runs on Gradescope (not just on your computer). Your files must be named correctly, and you must follow all instructions carefully to ensure that everything runs correctly on Gradescope.

A mark of zero (0) will be given for hardcoding results or other attempts to fool the autograder. Your code must work for **any** test cases.

The weight of this assignment is 9% of the course mark.