CMPUT 274/5 - Tangible Computing **Fall 2019**
Weekly Exercise #1

Password Validator

A secure password should ideally be long and contain a random mixture of different character types. For example, the password "Au7%j!mlO?" will take much longer to hack (and will be more secure) than a simpler password like "HUNTING".

In this exercise, you will write a program that will act as a password security checker (Part 1) and a secure password generator (Part 2). First, you will take a password and determine if it is Secure, Insecure, or Invalid, according to some simple criteria. Then, you will pseudo-randomly generate your own Secure passwords.

Note: Although we classify passwords as "Secure", "Insecure", or "Invalid" in this exercise, it is important to note that these are not robust definitions. For example, "Passw0rd!" would be considered secure under our system, even though this is a fairly weak password by most standards. Among other issues, our program does not check if the entered password contains English words (making it easier to crack) and should therefore not be used to choose a password for a sensitive application.

Your Task: Part 1, Password Validation

In a file called `validator.py`, write a function called `validate(password)` which will take a password as an argument, analyze it, and return either "Secure", "Insecure", or "Invalid".

```
def validate(password):
    """ Analyzes an input password to determine if it is "Secure", "
   Insecure", or "Invalid" based on the assignment description criteria.

    Arguments:
        password (string): a string of characters

    Returns:
        result (string): either "Secure", "Insecure", or "Invalid".
    """
    pass
```

Here are the criteria for each type of password:

1. **Invalid:** A password is considered invalid if:
   (a) it contains one or more "forbidden characters". These characters include the space character (" "), the hyphen ("-"), and the underscore ("_").
   (b) It is less than 8 characters in length.
2. **Insecure:** A password is considered insecure (valid but weak) if:
   (a) It is not invalid (It has 8 or more characters and does not contain any forbidden characters).

(b) It is not secure (it violates one or more of the conditions for a secure password).

3. **Secure:** A password is considered secure (valid and strong) if:

   (a) It is not invalid (It has 8 or more characters and does not contain any forbidden characters).

   (b) It contains at least one of all of the following character types:

      i. Uppercase characters (A..Z)

      ii. Lowercase characters (a..z)

      iii. Decimal digits (0..9)

      iv. Special characters which for the purposes of this program are all of the characters in the following string (enclosed by double quotes). You can copy the special characters as a plaintext string from this eClass link. Do not try to copy them from the PDF, as this may result in formatting errors.

      `!#$%&'()*+,./:;<=>?@[]^‘{|}~`

      Guarantee: Note that you will never see a double quotation mark (") or a backslash (\). This is to make this string of special characters easier to store in your Python program.

Here are a few sample test cases:

Note: Your function must take the following Inputs as **arguments**. Do not call the `input` function from within `validate`! Furthermore, your functions must **return values**, and should not print anything directly using `print`. To test these inputs and outputs, you should use the following lines of code **view the outputs in the terminal directly**:

```
if __name__ == "__main__":
    print(validate(INPUT_STRING))
```

where INPUT_STRING is the password passed as an argument to your `validate` function. For example, the first test case below could be called using:

```
if __name__ == "__main__":
    print(validate("HACKING"))
```

Note that if you remove the call to print in the above snippet (ie, `validate("HACKING")`), the code should produce *no output* as you should not print anything from within the functions themselves.

```
Input #1: HACKING
```

```
Returns: Invalid
```

Explanation: This password is too short (only 7 characters) so it is not valid.

```
Input #2: Passw0rd!
```

```
Returns: Secure
```

Explanation: This password contains at least one uppercase and one lowercase letter, one digit, and one special character. It is also at least 8 letters long, so it is secure.

```
Input #3: helloworld!
```

```
Output #3: Insecure
```

Explanation: This password does not contain any uppercase characters or digits, so it is not secure.

Your Task: Part 2, Password Generation

In the second part of the assignment, you will write a second function in `validator.py` called `generate(n)`. This function takes a length `n` and pseudo-randomly generates a password with `n` characters which is guaranteed to be Secure according to the criteria from Part 1.

It is up to you to determine a way to generate a password with this guarantee. Some implementations may be considered better (more efficient or more uniformly random) than others. For this reason, a correct solution will **not** receive full marks if it is extremely slow, inefficient, or predictable. Requirements:

- Your function **must pseudo-randomly generate** each password. You may want to use the `random` module.
- The password returned must be of length `n` (where `n` >= 8).
- Note that the passwords you return must **always** be Secure. You must find a way to guarantee this for every password you generate.

```
def generate(n):
    """ Generates a password of length n which is guaranteed to be Secure
   according to the given criteria.

    Arguments:
        n (integer): the length of the password to generate, n >= 8.

    Returns:
        secure_password (string): a Secure password of length n.
    """
    pass
```

Submission Guidelines:

Submit all of the following files as a compressed archive called `password_validator.tar.gz` or `password_validator.zip`:

- `validator.py` (template available on eClass, containing your implementation of the functions `validate` and `generate`.
- your `README`, which must follow the specifications in the Code Submission Guidelines.

Note that your files and functions must be named **exactly** as specified above.