University of Western Ontario, Computer Science Department
CS1026A SU19, Computer Organization

Assignment 2 **Due: May 24, 2019**

General Instructions: This assignment consists of 3 pages, 1 exercise, and is marked out of 100. Assignments are the independent work of each student. Software may be used to detect cheating.

Non-Functional Code Instructions:

1. Include brief comments in your code. Identifying yourself (the code's author) by name and user ID in the initial comment header. Comment also on key instructions and calculations in your code.
   e.g.
   ```
   ##
   # A program for computing areas with inscribed circles.
   # Student Name:  Alex Brandt
   # Student ID: abrandt5
   ```

2. Follow *good coding style* and normal Python conventions. This includes, but is not limited to:
   (i) Meaningful variable names.
   (ii) Conventions for naming variables and constants.
   (iii) Use of constants over "magic numbers".
   (iv) *Readability*: indentation, appropriate white space (blank spaces) within instructions, consistency in the use of all of the above.

Evaluation:

1. Functional Requirements:
   (i) Does your module correctly implement the four validation functions?
   (ii) Does your main program behave according to the specifications?
   (iii) Does your main program handle invalid input?
   (iv) Does your main program output values according to the specifications?

2. Non-Functional Requirements (above).
3. Ability to follow directions precisely.

Submission Instructions: Your submission should include exactly two files (zipped into a zip file, if you'd like). These two files are: `stringvalidation.py` and `userid_main.py` where userid is replaced by your UWO User ID (everything preceding "@" in your UWO email; e.g. abrandt5). The contents of these two files are described below.

Learning Outcomes: In this assignment we will look at using loops, functions, and user-defined modules to facilitate some basic string processing.

Exercise 1. This problem deals with *form validation*. Anyone who has created accounts online or made purchases online should be aware of form validation. This process is all about checking if the data the user has entered matches some pre-defined pattern. For example, a credit card number should be 16 digits long, possibly with a hyphen (-) between each group of 4 digits. In this exercise we look to define several different functions for validating several different kinds of input. The guidelines for accomplishing this task are broken into two parts.

Part 1. In this section we will define the contents of the `stringvalidation.py` file. In this file we look to implement four functions: `checkName`, `checkEmail`, `checkPassword`, and `checkAddress`. Each function follows a common scheme:

1. Each function takes one string parameter as input.
2. Each function returns a Boolean value to indicate if the input string is "valid". The functions returns `True` if the input is valid and `False` otherwise.
3. If an input string is invalid, the function prints a message to indicate *why* the string is invalid.

For each function, what determines if a string is "valid" depends on the particular function. The following criteria certainly do not define *all* possible valid, for example, emails, but you should limit your validation to only the following criteria.

1. The function `checkName` determines if the input string holds a valid first name and last name based on the following criteria:
   (a) The string must be at least 1 *alpha* character (i.e. a-z or A-Z) followed by a blank space character ( ) followed by at least 1 *alpha* character,
   (b) The first character of the string must be upper case,
   (c) The first character *after* the blank space must be upper case,
   (d) Hyphens (-) are valid as long as there are *alpha* characters to both the left and right of each hyphen.
2. The function `checkEmail` determines if the input string is a valid email address based on the following criteria:
   (a) The string begins with some positive number of alpha characters or numeric characters (0-9), followed by
   (b) The @ character, followed by
   (c) Some number of alpha characters, followed by
   (d) A period (.), followed by
   (e) One of: "com", "net", "org", "ca".
3. The function `checkPassword` determines if the input string is a valid password based on the following criteria:
   (a) There is at least one lower-case letter,

(b) There is at least one upper-case letter,

(c) There is at least one number,

(d) There are no spaces,

(e) The password is at least 8 characters long.

4. The function `checkAddress` determines if the input string is a valid address based on the following criteria:

   (a) The string begins with one or more numeric characters (0-9), followed by

   (b) A black space character ( ), followed by

   (c) One or more words containing only *alpha* characters (i.e. a-z or A-Z, there are no numbers or special characters).

Part 2. In this section we define the contents of the `userid_main.py` file. In this file you shall `import` your other file (e.g. by the command `from stringvalidation import checkName, checkEmail, checkPassword, checkAddress`) and then use those imported functions within your `main` function to prompt the user for inputs continuously until all inputs are validated. In particular, your main function should:

1. Prompt the user for their first and last name and store this value in a variable. Using the `checkName` function, determine if the input name is valid. If not, prompt the user again to enter their name again. Continue this process until a valid name is obtained.

2. Prompt the user to input a valid email address and store this value in a variable. Using the `checkEmail` function, determine if the input email is valid. If not, prompt the user again to enter another valid email address. Continue this process until a valid email address is obtained.

3. Prompt the user to input a valid password and store this value in a variable. Using the `checkPassword` function, determine if the input password is valid. If not, prompt the user again to enter another password. Continue this process until a valid password is obtained.

4. Prompt the user to input a valid street address and store this value in a variable. Using the `checkAddress` function, determine if the input address is valid. If not, prompt the user again to enter another street address. Continue this process until a valid street address is obtained.

5. Once all four inputs are validated, print the following message to the user:

   ```
   Thank you, <name>.
   Your email is:  <email>
   Your password is:  <password>
   Your address is:  <address>
   ```

   Where each of `<name>`, `<email>`, `<password>`, and `<address>` are replaced by the validated strings which the user has input.