# Due: Friday Oct. 6th, 2017 at 3 PM

**Assignment Guidelines**

- This assignment covers material in Module 03.
- Be sure to use strings that exactly match those specified on the assignment. Some of the used strings are provided in the interface provided for this assignment.
- You may use any string method
- Submission details:
  - Solutions to these questions must be placed in files `a03q1.py`, `a03q2.py`, `a03q3.py`, and `a03q4.py`.
  - You must be using Python 3 or higher. Do NOT use Python 2.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page
- Download the testing module from the course web page. Include `import check` in each solution file.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use Python constructs from later modules (e.g. loops and lists). Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - While you may use global *constants* in your solutions, do **not** use global *variables* for anything other than testing.
  - Read each question carefully for additional restrictions or tips.
  - Tip: for this assignment, repetition can be implemented using recursion as loops are not allowed

**The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.**

1. Write a Python function `samechars` that consumes two strings (`s1, s2`), returns `True` if all characters in s2 exist in s1 in the same order but not necessarily consecutively, otherwise the function returns `False`.
   For example:

```
samechars("", "") => True
samechars("acdbba","ada") => True
samechars("acdbba","abd") => False
```

   **Note: You may NOT use any of the `str` methods (such as `find`, etc.), however you may use `==,` and `slicing`.**

2. Write a Python function `generate_username` that consumes nothing and will prompt the user for three pieces of information (first name, last name, and birth year) and returns a username (string) according to the description below (see examples for clarification). The username will be made up from the first letter from the first name (in lowercase) followed by the last four letters from the last name (if exist, in lowercase, followed by the first three digits (from the left) of the birth year reversed. You may assume that each individual user input will be nonempty and that the user will enter 4-digit year.
   The prompts to be used should be (in the following order):

```
Enter your first name:
Enter your last name:
Enter your birth year:
```

   A few examples when run through Python would be:

```
>>> u=generate_username()
    Enter your first name:Mike
    Enter your last name:Brown
    Enter your birth year:1989
>>> u
    'mrown891'
>>> generate_username()
    Enter your first name:Young
    Enter your last name:Hu
    Enter your birth year:1991
    'yhu991'
>>>
```

3. The "strength" of a password is related to how difficult it would be to guess by either a person or by software. In this question we will set rules to reject/accept a password and to determine its strength.

*Acceptance rules*: accept the password if the password includes:

• At least one uppercase alphabetic character (A, B, C, …, Z), and

• At least one lowercase alphabetic character (a, b, c, …, z), and

• At least one digit (0, 1, 2, …, 9)

*Score calculation*: The score of a password starts at 0, and is adjusted as follows:

• Deduct 10 points if its length is less than 5;

• Add 15 points if its length in more than 8;

• Don't add points if it contains at most one "special" character, i.e. a character which is neither uppercase/lowercase alphabetic character, nor digit.

• Add 10 points for each additional "special" character (other than the first "special" one), e.g., if the password contains 3 "special" characters, add 20 points.

*Strength determination*: The strength of a password is:

• "weak": if its score is less than 25

• "medium": if its score is at least 25 but no more than 40

• "strong": if its score is more than 40

Write a python function `password_check` that consumes a string, named `password`, and returns `False` and **prints** formatted message (as described in the examples below) if `password` is rejected. Otherwise, the function returns a string formatted as `"score:strength"` (see examples below).

For example:

- `password_check("Xy 37 1-0") => "35:medium"`
- `password_check("Password999?") => "15:weak"`
- `password_check("Yt3)(*&a%") => "55:strong"`
- `password_check("PaSsWoRd") => False,` and prints
  `The password ("PaSsWoRd") failed a basic test`
- `password_check("hello12") => False,` and prints
  `The password ("hello12") failed a basic test`

Make sure that the function **returns and prints** the right format along with the right content

Hint: you might want to consider using helper function(s) for certain tasks (which some might be recursive)

4.

Note: For part (a) of Q4 you are NOT allowed to use any helper function

a) Write a Python function `draw` that consumes a string, `s`, (of length 1) and `b`, a positive natural number, and prints a shape as follows

first line: `s` printed `b` times

second line: `s` printed `b-1` times

….

$b^{th}$ line: `s` printed once

next line: `=` printed `10` times

then the first `b` lines printed again in reverse order

Please see the provided examples below

`draw("*", 6)` prints:

```
******
*****
****
***
**
*
==========
*
**
***
****
*****
******
```

`draw("$", 1)` prints:

```
$
==========
$
```

Note: For part (a) of Q4 you are NOT allowed to use any helper function

b) Write a Python function `draw_diamond` that consumes a natural number (`rows`), and prints a diamond of `"$"` characters of height (`2*rows-1`) as shown in the examples.

A few examples when run through Python would be:

```
>>>
>>> draw_diamond(0)
>>> draw_diamond(1)
$
>>> draw_diamond(5)
    $
   $$$
  $$$$$
 $$$$$$$
$$$$$$$$$
 $$$$$$$
  $$$$$
   $$$
    $
>>> draw_diamond(8)
       $
      $$$
     $$$$$
    $$$$$$$
   $$$$$$$$$
  $$$$$$$$$$$
 $$$$$$$$$$$$$
$$$$$$$$$$$$$$$
 $$$$$$$$$$$$$
  $$$$$$$$$$$
   $$$$$$$$$
    $$$$$$$
     $$$$$
      $$$
       $
>>>
```

Note: there are no spaces printed after the last $ in each line.