CS 116 Winter 2015

Assignment 06

Due at Noon on Wednesday, March 11 – Updated: March 4

Assignment Guidelines:

- **This assignment consists mostly of material covered in Module 06.**
- **You may use recursion or abstract list functions in your solutions. Do not use loops (from Module 08).**
- **When a function prints to the screen, your output must match the described output correctly, or you will lose correctness marks.**
- Your solutions should be placed in files **a06qY.py**, where Y is a value from 1 to 4.
- Download the testing module from the course web page. Include `import check` in each solution file.
    - When a function produces a floating point value, you must use `check.within` for your testing. Unless told otherwise, you may use a tolerance of 0.0001 in your tests.
- Note the following new restrictions:
    - Do not import any other modules other than `math` and `check`.
    - Do not define helper functions locally.
    - Examples and tests are not required for any helper functions.
    - Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post (#19) on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
    - You may use global constants in your solutions.
    - Do not use global variables for anything other than testing.
- Download the interface file from the course Web page to ensure that all function and structure names are spelled correctly and each function has the correct number and order of parameters. Use the function headers and full structure definitions in your submitted files for each question.
- For full style marks, your program must follow the **Python section** of the CS116 Style Guide. In particular,
    - Be sure to include all the steps of the design recipe for all required functions: including purpose statements, effects, contracts (including requirements), examples (note the new style), and tests.
    - You are not required to submit any templates with your solutions.
    - The purpose should be written in your own words and must include meaningful use of the function's parameter names.
    - There will be marks assigned for the appropriate use of constants, helper functions, choice of meaningful names, and appropriate use of whitespace.
- 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.
- Do not send any code files by email to your instructors or ISAs. It will not be accepted by course staff as an assignment submission, even if you are having issues with MarkUs. Course staff will not debug code emailed to them.
- Test data for all questions will always meet the stated assumptions for consumed values.
- Read each question carefully for specific restrictions.
- No late assignments will be accepted.
- Check MarkUs and your basic test results to ensure that your files were properly submitted. In particular:
    - A misnamed file or function will receive no marks.
    - Do not copy any text from the interactions window in WingIDE or from this pdf into your programs (even as comments). It will make your submitted file unreadable in MarkUs and you will receive no marks (correctness or style) for that question.
- Be sure to review the Academic Integrity policy on the Assignments page.

Coverage: Module 06
Language level: Python 3

Assignment 06

Due at Noon on Wednesday, March 11 – Updated: March 4

1. The "strength" of a password is related to how difficult it would be to guess (by either a person or by software). Some sites will reject a password if, for example, it contains only digits or if it is too short. Other sites will calculate a numeric grade (or strength) of a possible password based on properties of the string. In this question you will combine these two approaches, as described below.

   Rejection Rules: Reject a password if any of the following conditions are satisfied:
   - The password is an empty string;
   - The password when converted to lowercase is "password";
   - All characters in the password are alphabetic and all are uppercase letters;
   - All characters in the password are alphabetic and all are lowercase letters;
   - All characters in the password are digits.

   Score Calculation: The score of a password starts at 0, and is adjusted as follows:
   - Deduct 10 points if it contains fewer than 5 characters;
   - Add 10 points if it contains more than 8 characters;
   - Add 10 points if it contains at least one uppercase character, at least one lowercase character, and at least one digit;
   - Add 20 point if it contains at least one "special" character, i.e. a character which is not uppercase, lowercase, or a digit;
   - Add 5 points for each additional "special" character (other than the first "special" one) which is not uppercase, lowercase, or a digit, e.g. if the password contains 3 "special" characters, add 10 points.

   Write a Python function `password_strength` which consumes a string, called `password`, and returns `False` and prints a formatted message (as described in the examples below) if `password` is rejected by any of the rejection rules above. Otherwise, `password_strength` produces the numeric score for `password`, using the score calculation rules above.

   For example:
   - `password_strength("Xy 37 1-0") => 50`
   - `password_strength("Password999?") => 40`
   - `password_strength("Aaaa") => -10`
   - `password_strength("PaSsWoRd") => False`, and prints
   `The password ("PaSsWoRd") failed a basic test`
   - `password_strength("hello")=> False`, and prints
   `The password ("hello") failed a basic test`

   The displayed message will always include the value of `password` (with double quotes) in brackets, as illustrated.

   Note: As usual, you may use any of the functions listed in Piazza Post @19. Some of the string operations will greatly simplify your solution. Remember to use the `help` function to learn more about these operations.

Assignment 06

Due at Noon on Wednesday, March 11 – Updated: March 4

2. The game 6 Nimmt! is played with a deck of 104 cards numbered 1-104, inclusive. Each card has a point value determined by the number on the card:
   - 55 is worth 7 points;
   - multiples of 11 (other than 55) are worth 5 points;
   - multiples of 10 are worth 3 points;
   - multiples of 5 (other than multiples of 10 and 55) are worth 2 points;
   - any other card is worth 1 point.

   Write a Python function `points_6nimmt` that consumes `cards`, a list of distinct natural numbers between 1 and 104, inclusive, and produces the total number of points earned by `cards`. For example,
   - `points_6nimmt([4,2,100,55,44, 21]) => 18`

3. In Piazza, it is possible to search through posts for keywords. For the purposes of this question, we will represent a Piazza post (containing the post number, its title, its body, and the instructor's reply) as a single string with the following format: `"@N::Title::Body::Reply"`. For example,
   - `"@507::empty in Q1/Q2::can list be empty::yes"` corresponds to post @507, with the title `"empty in Q1/Q2"`, the body `"can list be empty?"`, and the reply `"yes"`
   - `"@19::Allowable functions::empty?, first, rest::"` corresponds to post @19, with title `"Allowable functions"`, the body `"empty?, first, rest"`, and no reply.

   You may assume that `N` > 0, `Title` and `Body` are non-empty, but note that `Reply` may be empty.

   Write a Python function `search_piazza`, that consumes `posts`, a list of strings in the format described above, and `keyword`, a nonempty string, and produces a list of Nat, containing the post numbers of every string in `posts` which contains `keyword` as a substring (as determined by the operation `in`). For example, suppose `cs116` contains:
   ```
   [ "@507::empty in Q1/Q2::can list be empty::yes",
     "@1::CS116 W2015::Welcome to first post::Thanks!",
     "@19::Allowable functions::empty?, first, rest::"]
   ```
   then
   - `search_piazza (cs116, "empty") => [507, 19]`
   - `search_piazza (cs116, "Welcome") => [1]`
   - `search_piazza (cs116, "First") => []`
   - `search_piazza (cs116, "1") => [507,1,19]`

   The produced list should be in the same relative order as `posts`.

Assignment 06

Due at Noon on Wednesday, March 11 – Updated: March 4

4. The "board" in 6 Nimmt! consists of 4 rows of cards. Each row is in increasing order and contains between 1 and 5 cards, as shown. Players take turn placing new cards on the board, one at a time. New cards can only be placed at the rightmost end of a row. In the process of placing cards, a player may need to take all the cards in a row, and will be assigned points based on the values in that row. The goal of the game is to have the lowest score after all rounds have been played.

To determine where a card is to be placed, you need to examine the rightmost card in each row (70,19,8,93 in the board on the left). A card cannot be placed next to a bigger card, and must be placed next to the card its value is closest to. For example,

- 17 must be placed in the third row, next to 8;
- 22 must be placed in the second row, next to 19;
- 89 must be placed in the first row, next to 70.

There are three possibilities:

1. The card can be placed, using the described guidelines, in a row with fewer than 5 cards.
2. If the row already has 5 cards, the new card cannot be added. Instead, the new card must actually replace the entire row, and the player takes all the cards in that row. Consider, for example, placing 99. Based on the numbers, it must go in the fourth row. However, since that row already has 5 cards, the player will take the cards there (13,30,50,88,93) and then 99 will become the only card in the fourth row.
3. If the new card is smaller than all the rightmost cards, then the player can choose to take any one of the rows, and the new card then becomes the only card in that row. For example, 5 cannot be played at the rightmost end of any row, so the player would get to choose which row to take, and 5 would then become the only card in that row. Typically, a player chooses the row containing the fewest number of points, but can choose any row.

Consider the following new data definitions:

```
## A Card_6Nimmt is a Nat between 1 and 104, inclusive.
## A Row_6Nimmt is a (listof Card_6Nimmt) containing between 1 and 5 values,
##   in strictly increasing order.
## A Board_6Nimmt is a (listof Row_6Nimmt) of length 4. There are
##   no duplicates between any of the Row_6Nimmt lists in the board.
```

So, for example, the 6 Nimmt! board in our image would be represented as in `my_board`:

```
my_board = [[67,70], [9,18,19], [8], [13,30,50,88,93]]
```

Assignment 06

Due at Noon on Wednesday, March 11 – Updated: March 4

Write a Python function `turn_6nimmt`, that consumes `board`, of type `Board_6Nimmt`, and `card`, of type `Card_6Nimmt`, and mutates `board` to reflect the playing of `card` (as described previously). In addition,

- if `card` is played on `board` without the player taking any cards (as in possibility 1) then `True` is produced;
- if `card` should be played on a row which already contains five cards (as in possibility 2), then a row containing `card` replaces that row, and the function produces the contents of the taken row;
- if `card` is smaller than the rightmost card in each row of `board` (as in possibility 3), then the program will print a prompt with each row (as described in the examples below) and ask the player to indicate which row to take; that row is then replaced by a row containing `card`, and the function produces the taken row.

For example, consider the following sequence of calls (in which the value of `my_board` is being updated by each successive call, as indicated.

`turn_6nimmt(my_board, 17) => True`, and mutates `my_board` to
`[[67, 70], [9, 18, 19], [8, 17], [13, 30, 50, 88, 93]]`

`turn_6nimmt(my_board, 22) => True`, and mutates `my_board` to
`[[67, 70], [9, 18, 19, 22], [8, 17], [13, 30, 50, 88, 93]]`

`turn_6nimmt(my_board, 99) => [13, 30, 50, 88, 93]`, and mutates `my_board` to
`[[67, 70], [9, 18, 19, 22], [8, 17], [99]]`

`turn_6nimmt(my_board, 5)` prints board information and prompts the player for a choice:

```
Enter 1 to take [67, 70]
Enter 2 to take [9, 18, 19, 22]
Enter 3 to take [8, 17]
Enter 4 to take [99]
Enter choice: 3
```

After the player enters *3*, the function produces `[8, 17]`, and `my_board` is mutated to
`[[67, 70], [9, 18, 19, 22], [5], [99]]`

Note: when printing the information for the player to choose `row`, use `print(row)` to print a particular `row` in the format indicated, being careful to keep the prompt and the row information on the same line.