ENGI 1006 HW2 (Chapters 3, 5)

Assignment Due on 10/13/2016 at 11:59PM

Note: you can only use up to 48 hours of your total grace period for this assignment. This allows us to release the solutions on 10/15 before the midterm on 10/19.

Academic Honesty Policy:
You are expected to submit **your own work**. Assignments are to be completed individually, NOT in groups. No collaboration is permitted. Please do not include any of your code snippets or algorithms in public Piazza posts. You cannot not use solutions from any external source. If you have any questions at all about this policy please ask the course staff before submitting an assignment.

For this, and all future homeworks, we'll have one main Piazza post for the homework overall, as well as more specific posts for each homework question.
General questions should be asked as follow-up discussions beneath the main Piazza post: https://piazza.com/class/isq8q3q8iev6dp?cid=90

Please note that we will be modifying this main Piazza post with clarifications on the homework where necessary, and you will be responsible for adhering to them.

There are links to Piazza posts for individual questions below. If you have a question, please post a follow-up discussion *on the appropriate post.* Any relevant discussions will be incorporated as a clarification in the main post.

For all problems:

- **A skeleton of each function has been provided for you. DO NOT modify the function signatures or return variables for any reason. Your job is to complete the body of each function. We've included test cases along with expected output so you can check your solutions. These test cases are under the line:**

```
if __name__ == "__main__":
```

  You can modify these test cases within the skeleton code.

- **Use Python 3.5 for all Python parts. To check your Python version, enter "python --version" (no quotes) in a terminal.**

- **Make a comment at the top of each file explaining the high-level logic of your code. Make in-line comments detailing specific decisions and logic where necessary.**

- **You will be graded on your code style. Please refer to the Python style guide available on Courseworks in the resources folder.**

- **Download the skeletons from Canvas and keep names the same:**

- **piri.py**
- **cross_sort.py**
- **reverse_list.py**
- **missing_letters.py**
- **palindrome.py**
- **is_strong_password.py**
- **string_eval.py**

For submitting, put all seven files into a single folder named UNI_hw2. Compress your UNI_hw2 folder into a zip file named UNI_hw2.zip (eg. abc1234_hw2.zip). Make sure to use an underline, NOT a hyphen. Submit that zip file via courseworks.

Function **type annotations** are included within these instructions to concisely explain expected parameters and return values. For example:

```
cross_sort(l: list[int]) → list[int]
```

means that the function `cross_sort` takes a parameter `l`, which is going to be a list composed of ints, and that the function will return a list composed of ints.

1. (25 pts) **`def piri() -> None`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=83

iOS 10 was released a few weeks ago. It includes a function called Siri, an intelligent personal assistant to whom you can talk and ask questions. With Python, you can make your *own* Siri! Instead of voice recognition, we will communicate with Piri (Python-Siri) through the console.

Complete the function piri(), that takes no arguments, along with all of its helper functions, which do take arguments (see below). It first prompts the user for their name, then prompts the user to engage in one of four interactions which can be selected by inputting the number corresponding to the interaction he/she wants:

1) Give me a random number between X and Y
2) What is my BMR?
3) Tell me if this number is prime.
4) Exit

You must call the helper functions within the piri() function to provide the functionality for the interactions.

The general flow here is:

- piri() is called once
- It indefinitely prompts the user to choose an interaction, stopping only after the user specifies that they would like to Exit
- Every time an interaction is chosen, piri collects input that is specific to that interaction
    - The input collected for that interaction is parsed into variables
    - Those variables are used to call the function corresponding to the interaction that the user specified
    - The output of the function is collected and printed as specified below

Here is an example interaction:

$ python piri.py
>> Hi, I am your personal assistant Piri. What is your name?
Alex
>> Hi, Alex! What can I do for you? Select from:
1) Give me a random number between X and Y
2) What is my BMR?
3) Tell me if this number is prime.
4) Exit
1
>> Between which numbers would you like me to pick? 3 10
7 is a random number between 3 and 10.

>> Hi, Alex! What can I do for you? Select from:

1) Give me a random number between X and Y

2) What is my BMR?

3) Tell me if this number is prime.

4) Exit

4

Have a good day, Alex!

a) "Give me a random number between X and Y: "

```
def bound_random(x: int, y: int) → int
```

Complete this helper function. The random integer Piri returns should be within X inclusive, Y not inclusive. Assume that the inputs X and Y will satisfy X < Y. *Hint: you may want to use the random module.*

>> Between which numbers would you like me to pick? 3 10

7 is a random number between 3 and 10.

Note that, when prompted, the user should enter both X and Y on the same line, with a single space separating X from Y. You must parse that input into the two ints that will become the inputs for bound_random.

b) "What is my BMR?" (Basal Metabolic Rate)

```
def bmr(weight: int, height: int, age: int, gender: str) → float
```

Complete this helper function. Piri should ask and wait for user's input on each question:

>> What is your weight in pounds? 160

What is your height in inches? 70

What is your age? 20

"m" or "f"? m

Your BMR is 1815.80

In case of invalid input for one of those questions, Piri should repeat the question until the input is valid. Base BMR on this formula (use the English BMR formula). BMR should round to two decimal places.

c) "Tell me if this number is prime."

```
def prime(x: int) → bool:
```

It should print:

>> Which number? 4

4 is not prime

You should appropriately handle negatives and floats. Feel free to reuse code from homework 1.

d) “Exit”

Have a good day, Alex!

And then the program will return.

e) If user inputs anything not covered in the above cases, Piri should print the following line:

I’m sorry, I don’t understand.

After each command, Piri should prompt the user the same way it did in the beginning.

2. (20 pts) **`def cross_sort(L: list[int]) -> list[int]`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=84

Write a python function cross_sort(L) that takes an unsorted list of integers L and returns a new list composed of the original elements that fulfills the following requirements:

- Even-indexed elements (index 0,2,4…) from the original list are sorted in ascending order in the new list
- Odd-indexed elements (index 1,3,5…) from the original list are sorted in descending order in the new list

Note that the input list L should NOT be changed.

```
>> cross_sort([1,3,4,2,9,6,-3,0])
[-3,6,1,3,4,2,9,0]
```

3. (10pts) **`def reverse_list(L: List[Any]) -> List[Any]`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=85

Write a function reverse_list(L) which takes a list L and returns a new list composed of the elements of L in reverse order. You may not return L[::-1], use the built-in list.reverse() function, nor use the reverse() function.

>> reverse_list([1,2,3,5,7,8])
[8,7,5,3,2,1]

>> reverse_list([“i”,”love”,”1006”,”more”,”than”,”anyone”])
[“anyone”,”than”,”more”,”1006”,”love”,”i”]

4. (10 pts) **`def missing_letters(s: string) -> list[str]`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=86

Write a function missing_letters(s) that takes a string argument “s”, and checks if s contains every letter of the alphabet at least once. It will return a sorted list containing all the letters of the alphabet that are not present within “s”. The results should not depend on capitalization.
Hint: you may want to look at the chr(), ord(), and lower() functions.

>> missing_letters(“The quick brown fox jumps over the lazy dog.”)
[ ]

>> missing_letters(“The brown fox jumps over the dog”)
[‘a’,’c’,’i’,’k’,’l’,’q’,’y’,’z’]

5. (15 pts) **`def palindrome(s: str) → int`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=87

Palindrome - https://en.wikipedia.org/wiki/Palindrome
Given a non-empty string, return the minimum number of characters that could be replaced by some other character in order to make it a palindrome. The function will return this number of characters as an int. You should ignore capitalization, treating capital letters the same as lower case letters.

Examples (the explanation will not be required in the actual program):

>> palindrome(“abccb”)
2
(Explanation: You would need to change it to one of: “abcba”, “bbcbb”, “accca”, etc.)

>> palindrome(“bDaadc”)
1
(Explanation: You would need to change it to “bDaadb” or “cDaadc”)

>> palindrome(“aBa”)
0
(Explanation: It is already a palindrome)

6. (20 pts) **`def is_strong_password(password: string) → bool`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=88

In 2016, information security should be one of your highest priorities. To protect your personal information from identity theft, a strong password is necessary.

A password is considered strong if all the following requirements are met:

- The password must be at least 6 characters and at most 20 characters
- It must contain at least one lowercase letter, at least one uppercase letter, and at least one number
- It must NOT contain a character that is repeated consecutively more than twice at any point. A password that contains “111” anywhere within it would be weak, but a password like “...11….1….11” would be strong, assuming other requirements are met within the “...”

Write a function, is_strong_password(password), that takes a string argument “password” and checks if “password” meets all of the strong requirements. If it does, return True. Otherwise, return False.

>> is_strong_password(“1a9fj”)
False

>> is_strong_password(“js88Sjf99”)
True

>> is_strong_password(“ys9aS890sFFFFn”)
False

7. [BONUS] 10 pts: **`def string_eval(expr: str) → int`**
Piazza: https://piazza.com/class/isq8q3q8iev6dp?cid=89

Write a function **string_eval**, that takes a string containing a valid arithmetic expression containing only the “+”, “-”, and “*” operators and non-negative operands. No intermediate result will be negative. There will be no parentheses nor any spaces. The function will evaluate the expression following the correct order of operations from left to right, and return the result. **You cannot use the eval() function.** You may find the find() and isdigit() functions useful.

```
>> string_eval("2+3*4-8")
6

>> string_eval(“5-3+1”)
3
```