15-110 Fall 2019
Hw 06

Out: Monday 21st October, 2019 at 18:00 AST
Due: Monday 28th October, 2019 at 18:00 AST

Introduction

In this homework you will practice with loops and function concepts.
The total number of points available from the questions is 175, 75 points are *bonus* points (i.e., you only need 100 points to get the maximum grade).
In your `.zip` file (see general instructions below), you need to include only the file `hw06.py` with the python functions answering the questions. In the handout you have found a file `hw06.py` with the functions already defined but with an empty body (or partially filled). You have to complete the body of each function with the code required to answer to the questions.

General Instructions for Submitting the Assignments

Submissions are handled through Autolab, at `https://autolab.andrew.cmu.edu/courses/15110q-f19`

You are advised to create on your computer/account a folder named `110-hw`. For each new homework, you should create a new sub-folder named `01`, `02`, etc. where you can put the files related to the homework. In this way you work will be nicely organized and information and files will be easily accessible.
You can also create an equivalent struture for the *laboratories*, where in this case the root folder should be named `110-lab`.

When you are ready with the homework and want to submit your solutions, you need to go in the current homework folder (e.g., `01`), *select all files you will submit* (that can include both `.pdf` files with written answers to questions and python code files, `.py`) and compress them in one single `.zip` file.[1]

According to the OS you are using, you migh have different options for making the zip file. For instance, on Windows, after selection of the files, you should right-click and select `Send to: Compressed folder`, while on macOS, you can select `Compress` on the menu appearing from the right-click.

The compression action will produce a zip file containing the files to be handed in for the assignment. The file should be named `hwXX-handin.zip` (e.g., for this homework, the name of the file should be `hw06-handin.zip`). Then, open Autolab, find the page for this assignment, and submit your `hw06-handin.zip` file via the "Submit" link.

☛ The number of submissions is limited to 5. The last submission is the one that will be graded.

Style

Part of your grade on assignments are style points, that can be lost if your code is too disorganized, unreadable or unnecessarily complicated. To avoid loosing style points, please follow the guidelines at `https://web2.qatar.cmu.edu/cs/15110/resources/style.pdf`.

[1] The (single) zip file is needed, even when the files handed for the assignment consists of one individual file.

1 Split the list

Problem 1.1: (15 points)

Implement the function `split_by_type(L)` that takes as input a list `L` that can include integer, floats, or strings, and returns a list of three lists, where the first sublist includes all the integers, the second all the floats, and the third all the strings, in the same order of appearance as in the original list.

For instance, if the list `['a', 1, 'c', 3.0, 4]` is given as an input, the function should return the following list of three lists: `[ [1,4], [3.0], ['a','c'] ]`. In case an element of the list has a different type than int, float, or string, the function should return the `'Error'` string.

2 Conversion from / to binary numbers

In our daily life, numbers are usually represented in *base 10*, meaning that a number is written using the numeric digits from 0 up to 10 (not included), $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, that are elevated to the powers of 10 based to their position. For instance, in base 10 the number 1982 is equivalent to write $1 \times 10^3 + 9 \times 10^2 + 8 \times 10^1 + 2 \times 10^0 = 1982$.

Instead, in a digital computer every number is represented as a binary number (*base 2*), meaning that a number is represented using only 0 and 1 digits elevated to the powers of 2 based on their position. For instance, in base 2 the number 10101 is equivalent to $1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 21$ (in base 10).

Check slides 19-22 in Lecture 8 (September 10) for more details about how to convert from a decimal number to a binary number and vice versa.

Problem 2.1: (15 points)

Implement the function `binary_to_decimal(b)` that takes a binary number `b` represented by a string of zeros and ones as input, and returns the integer corresponding to its decimal representation.

Problem 2.2: (15 points)

Implement the function `decimal_to_binary(d)` that takes an integer `d` as input, and returns the string of zeros and ones that is the binary representation of `d`.

3 DNA Representation

When DNA is stored in a computer, it can be represented as a list of letters. The letter A is used for adenine, C for cytosine, G for guanine, and T for thymine.

An important problem in working with DNA is *subsequence matching*. In short, subsequence matching determines if a short DNA sequence occurs within a longer sequence. (All the letters on the subsequence occur consecutively within the original sequence.) For example:

- The sequence `['A','C','C','C','C','T','T','T']` contains the subsequence `['C','C','T']` once.
- The sequence `['A','C','C','T','A','C','C','T']` contains the subsequence `['C','C','T']` twice.
- The sequence `['A','C','C','C','T','T','T','T']` contains the subsequence `['T','T']` three times.

Problem 3.1: (15 points)

Implement the function `simple_subseq_match(sequence, subseq)` which, given a DNA sequence `sequence` and a shorter DNA sequence `subseq`, returns how many times `subseq` occurs within `sequence`. You can assume that both `sequence` and `subseq` are lists containing only `'A'`, `'C'`, `'T'` or `'G'`.

Sometimes, when doing subsequence matching, we allow partial matches instead of perfect matches. A partial match is one where we do not care about the value of certain characters. If we do not care about a character, we give it the value `N`. For example, `['A', 'N', 'C']` can match any three letter sequence that starts with `A` and ends with `C`, such as `['A', 'T', 'C']`, `['A', 'G', 'C']`, etc.

Problem 3.2: (25 points)

Implement the function `subseq_match(sequence, subseq)` which, given a DNA sequence `sequence` and a shorter DNA sequence `subseq` which may contain `N` values, returns how many times `subseq` occurs within `sequence`. You can assume that both sequences are valid.

Here are some examples:

- `subseq_match(['A','C','C','T','C','T'], ['T','N','T'])` returns `1`.
- `subseq_match(['A','C','C','T','A','G','C','T'], ['N','C','T'])` returns `2`.
- `subseq_match(['A','C','C','C','T','T','T','T'], ['N','T','T'])` returns `3`.

4 Hangman game

Hangman is a game for two players where a player A chooses a *secret* word, and then player B tries to guess the secret word by picking one letter at a time. Each time the player B guesses a letter that is not in the secret word, he/she looses a turn. The word has to be guessed in at most n turns. A typical number for n is *7* (because, when played on paper, the turns correspond to drawing the head, body, two arms, two legs and hanging the hangman).

In this task you will implement a program for playing hangman with a friend. You might have noticed that there are no functions for this part of the homework in your started file. That is because this time you will implement the whole program yourself, from scratch! We will give you the instructions and you need to transform them into code. Read them carefully and make sure you understand. If you have any questions, do not hesitate to ask the course staff.

Problem 4.1: (2 points)

You need a function which you will call whenever you want to start a game. Let's call this function `hangman`. It should take one parameter called `nturns` representing the number of turns. This parameter has the default value 7.

For now, this function simply prints the string `"Welcome␣to␣Hangman!"`.

It is good practice to load and test your function always, even before it is completed. This avoids propagating errors and having to fix them all at the end.

Load the file and call the function to check if it works as expected.

Problem 4.2: (3 points)

The secret word must contain only characters from `"a"` to `"z"`, in lowercase. It cannot contain numbers, spaces or punctuation marks.

Implement a function `valid_word(w)` that takes a string and, if it contains only alphabetical characters, returns the word in lowercase. Otherwise, it returns `None`.

Test your `valid_word(w)` function.

Problem 4.3: (2 points)

Back to the `hangman` function, after printing the message, you need to prompt player *A* for the secret word[2].

Problem 4.4: (2 points)

You need to keep track of the state of the game as it proceeds. The game state includes the number of turns left, and a list of guessed letters. Declare two variables for that and initialize them with proper values.

[2]It seems there is no way to hide this word in Spyder. If you were executing the program in your computer's terminal, then it is possible.

Problem 4.5: (7 points)

Implement the function `all_guessed(L, W)` that takes as arguments a list of characters `L` and a string `W` and returns `True` if all the letters in `W` are in the list `L`.

Test your `all_guessed(L, W)` function on a few examples.

Problem 4.6: (7 points)

Implement the function `partial_secret(L, W)` that takes a list of characters `L` and a string `W` and returns another string which corresponds to `W` where the characters that are in `L` are used normally and the others are hidden by _. For example, `partial_secret(['a','o','n'], 'hangman')` should return:

```
_ a n _ _ a n
```

Add one space between the characters so that the string looks nicer.

Test your `partial_secret(L, W)` function.

Problem 4.7: (2 points)

Implement a function `valid_letter(c)` that takes whatever was typed by player B and, if it is a valid guess, returns the lowercase letter. Otherwise, it returns `None`.

Problem 4.8: (10 points)

Now it is time to go back to the `hangman` function and complete it. You will implement a loop that iterates for each letter player B guesses. Each time player B guesses a letter that is in the secret world, they can keep guessing another letter without loosing a turn.

At the beginning of the loop (or at the beginning of B's guess), print the list of letters guessed so far and the number of turns left.

After printing the game state information, prompt the user for a letter. Use your `valid_letter(c)` function to check if this is a valid letter and to get it in lowercase. In case it is not a valid letter, inform the player and ask for a letter again. Once you get a valid letter, if this letter is not in the secret word, one turn is spent.

At this point, you have a newly guessed letter and have decided if a turn was spend or not, so update the game state.

The loop must stop whenever there are no more turns left. If the loop finished, this means the player used all their turns and did not guess the word, so you can print a message indicating that they lost.

Now you can test your code. It should inform that you lost after trying out n letters that are not in the secret word, where n is the parameter passed to the `hangman` function.

Problem 4.9: (5 points)

Once the player chooses a letter, we need to update the list of guessed letters.

After this list is updated, print a string with the guessed letters visible and the other hidden by `\_`. You should use the helper function `partial_secret(L, W)`.

If you test your code again, at each guess the secret word is printed with the letters that were guessed correctly unveiled.

Problem 4.10: (5 points)

An important part of the game is still missing: the winning!

Inside the loop, each time a new letter is guessed, we need to check if all letters of the word were guessed. If so, print a winning message and `return` (the function can end right there and then). You should use the function `all_guessed(L, W)`.

Your game is done now! Here's an example of a winning and a loosing run[3]:

```
Welcome to Hangman!

Type the secret word you want your friend to guess: qatar

Guesses so far: []
Number of turns left: 7
Guess one letter: a
_ a _ a _

Guesses so far: ['a']
Number of turns left: 7
Guess one letter: t
_ a t a _

Guesses so far: ['a', 't']
Number of turns left: 7
Guess one letter: r
_ a t a r

Guesses so far: ['a', 't', 'r']
Number of turns left: 7
Guess one letter: z
_ a t a r

Guesses so far: ['a', 't', 'r', 'z']
Number of turns left: 6
```

[3]It is ok if your messages are different, as long as the game interaction is the same.

```
Guess one letter: q
q a t a r
Congratulations!

_


Welcome to Hangman!

Type the secret word you want your friend to guess: jazz

Guesses so far: []
Number of turns left: 7
Guess one letter: a
_ a _ _

Guesses so far: ['a']
Number of turns left: 7
Guess one letter: o
_ a _ _

Guesses so far: ['a', 'o']
Number of turns left: 6
Guess one letter: i
_ a _ _

Guesses so far: ['a', 'o', 'i']
Number of turns left: 5
Guess one letter: e
_ a _ _

Guesses so far: ['a', 'o', 'i', 'e']
Number of turns left: 4
Guess one letter: s
_ a _ _

Guesses so far: ['a', 'o', 'i', 'e', 's']
Number of turns left: 3
Guess one letter: b
_ a _ _

Guesses so far: ['a', 'o', 'i', 'e', 's', 'b']
Number of turns left: 2
Guess one letter: r
_ a _ _

Guesses so far: ['a', 'o', 'i', 'e', 's', 'b', 'r']
Number of turns left: 1
Guess one letter: t
_ a _ _

You lost :(
```

5 Find minimum in a list of lists

Problem 5.1: (20 points)

Implement the function `find_minimum(L)` that takes as input a list of heterogeneous elements, where each element can be an integer, a float, a string, a list of integers, or a list of strings. The function returns the minimum value among the elements of `L`. You can assume that the maximum value in the list cannot be higher than 100.

If none of the elements in `L` are numeric (i.e., either integer or float), `None` is returned.

The function makes use of a helper function `check_value(e)` that takes as input an object `e` and returns: `None` if `e` is a string or a list of strings; `e` if `e` is either an integer or a float; the minimum value in the list `e` if `e` is a list of numbers. You have to implement also the `check_value(e)` function.

For instance for `L = [1,3,6,-19, 7, [1, 0, -19.6], 'nn', [-7, -30], ['a', 'b'], 'hello']`, the call `find_minimum(L)` will return -30, while for `L = ['a', 'b', ['c', 'd']]` the returned value is `None`.

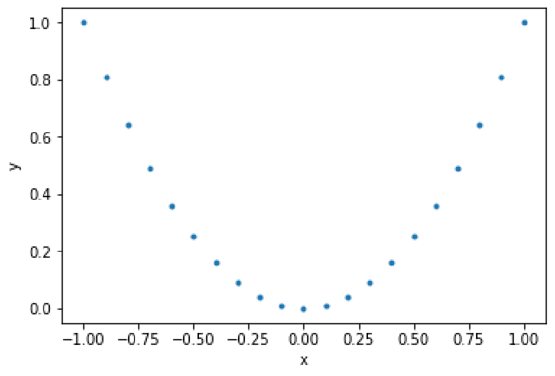
The function `find_minimum()` must include a *specification*, a docstring that describe the input, the output, and what the function does.

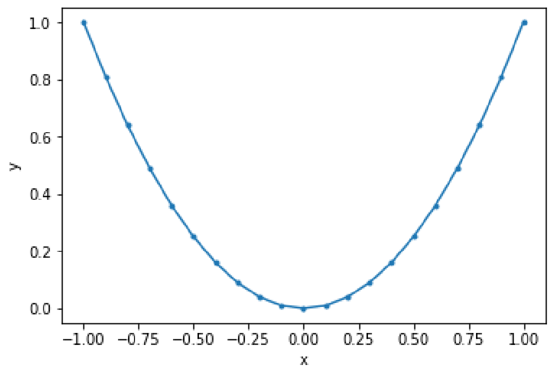
6 Plot a function

A *mathematical* function $f : X \to Y$ maps every point $x \in X$ to a point $y \in Y$, where X and Y are subsets of the real set, $\mathbb{R}$. The notation $y = f(x)$ indicates the value y that the function associates to the value x.

To plot the mathematical function $f(x)$ using a programming language (i.e, using programming functions), we need to select n points $x_i \in X$, $i = 1, \ldots, n$, compute the corresponding values $y_i = f(x_i)$, and plot all the n pairs (x_i, y_i), where each pair will correspond to a dot/point in the $X - Y$ Cartesian plane. We can also join the dots by a line, such that an overall smooth plot of the function will appear.

The figure below shows an example for the the function $f(x) = x^2$, where the figure to the left shows the function evaluated at the points from -1 to 1 (included) with a step of 0.1 (for a total of 21 points). The figure to the right shows the same thing but with the dots now being interpolated by lines.





Problem 6.1: (15 points)

Implement the function `plot_function(f, xmin, xmax, step, lines=True)` that takes as input a function object, `f`, that implements a mathematical function, three floats, `xmin`, `xmax`, `step`, and one optional boolean parameter, `lines`. The function plots the value of the mathematical function `f` for the values of x between `xmin` and `xmax` (included) every `step` points (as in the example above). The value of the optional parameter `lines` sets whether points are interpolated by lines or not, as it is explained below.

For plotting, `function_plot(f, xmin, xmax, step, lines=True)` makes use of the helper function `plot_points(x, y, lines=True)` that takes as input two lists of floats, `x` and `y`, that correspond to the (x_i, y_i) pairs to be plotted. The optional parameter `lines` sets the type of the plot: if `lines` is set to `True`, the plot shows interpolating lines, instead only dots are plotted when `line` is set to `False`.

In practice, in the function `plot_function()`, you have to generate the paired lists of x and y points and pass them to the helper function `plot_points()` that will manage their plotting. To generate the x and y lists of points you must use the `arange()` function from the `numpy` module, that allows to generate lists of real numbers.

The code of the helper function is the following (and it is provided in the handout):

```
def plot_points(x, y, line=True):
  plt.figure()
  plt.xlabel('x')
  plt.ylabel('y')
  if line:
      linestyle = '-'
  else:
      linestyle = ''
  plt.plot(x, y, marker='.', linestyle=linestyle)
```

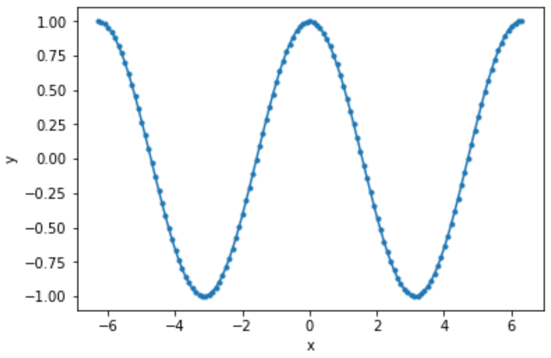
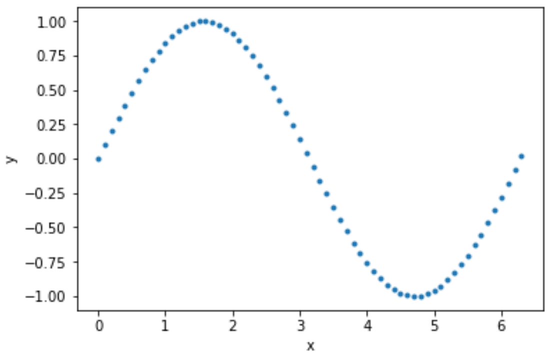
The helper function `plot_points()` makes use of four functions (`figure()`, `xlabel()`, `ylabel()`, `plot()`) from the module `matplotlib.pyplot`. In the code, the name of the module as been shortened and aliased to `plt`. You need to add, before the definition of `plot_points()`, the `import` statement that allows to correctly use the functions from the module as they are written.

Once you have written the function `function_plot()` and have added the required import statement, you can test the function and see the plotting graphs! You can test it using the functions provided by the `math` module (see lecture slides). Remember that in order to do this, you need to include the `import` statement for the module.

For instance, invoking the function as `plot_function(math.cos, -2 * math.pi, 2 * math.pi, 0.1)` and as `plot_function(math.sin, 0, 2 * math.pi, 0.1, False)` will produce the left and right plots shown in the figure below (`math.pi` is the value of π, as provided by the `math` module).

Problem 6.2: (10 points)

If you have successfully answered the previous question, now you have a practical tool to plot *any* mathematical function of interest. Remember that the first argument of `plot_function()` is a function object that implements a mathematical function. Therefore, let's build a little *library* of python functions each implementing a different mathematical function that can be displayed using `plot_function()`!

Implement the functions `parabola(x)`, `cubic(x)`, `periodic(x)` that realize, respectively, the mathematical functions $f(x) = 2x^2, f(x) = x^3 + 2, f(x) = 2\cos(4x)$. Each (pyhton) function takes as input a real value `x` and returns the value $f(x)$ that the mathematical function associates to x. For instance, `parabola(2)` returns 4, while `parabola(3.5)` returns 12.25.

For instance, invoking `plot_function(periodic, 0, 2 * math.pi, 0.05, True)` will produce the plot shown below.