Homework 1 Instructions

UCSD Extension CSE-41273, Spring 2019.

The zip download file has, besides this file, the file `HW1.py` with skeleton functions that you will implement for this homework. To start, be sure to read the Notes and Resources document, in Blackboard, under *Lessons -> Week 1 -> Notes and Resources*. Yes, it's a lot, because this is the start of the class. **First,** do the installation and setup of the development environment. In the Notes and Resources, there is a link to a file *About Homework*. **Read the *About Homework* file before attempting the homework!**

Please read and follow all the directions carefully! Read everything before coding!

When you are ready to turn in the file, use a zip utility to compress the file and rename the compressed file to `HW1_your_name.zip`, **as described in the *About Homework* document.**

Be sure to run `flake8` on your code before you turn it in! The `HW1.py` file should not have anything except the functions you implement. **If you have debugging print statements or test code, remove them before turning in the file.**

I have automated test programs that will check your code so there is no need to provide any "proof" that your code runs.

How to do the homework

Open a command window (Windows) or a terminal window (OSX/Linux), navigate to your class folder and **activate your virtual environment** so that when you run `python --version` on the command line it will report a Python 3 that is version 3.6 or above. Now when you type `python`, you should get the interactive REPL that has `>>>` as the prompt.

Open the file `HW1.py` in your code editor. Put your name in the appropriate comment at the top of the program file `HW1.py`. Edit the file to implement the functions according to the instructions below.

Docstrings - The lines that have """ strings following the function definition and at the beginning of the file are docstrings, which we will talk about more next week. **Do not move or remove the docstrings!** Put the implementation of the functions immediately **following** the docstring for each function. Then, to test the functions, *copy the code for the function to test* from the editor and paste it into the REPL and run some tests of the functions as shown in the "It should work like this" instructions for each one.

Except for where I tell you to look up useful methods, **everything you need to know is in the lessons or the supplemental required reading.** Don't waste your time searching for answers on the internet - I guarantee you will figure it out faster by reviewing the lesson and/or reading the documentation as instructed.

About the Python documentation: When the documentation shows, for example, `str.lower()`, the `str` part represents a string variable. So, if you had a variable `name` containing the string 'Diane Chen' and you wanted to make it all lower case, you would use `name.lower()`, which would return the string 'diane chen'.

Please note that all of your functions must return values! None of them print anything. If anything in the instructions is confusing, please email me with your questions.

Function 1: silly_case

Edit the function `silly_case` so that it takes a string and *returns a new string* where the new string is the same as the input string, but in *silly case*: each word has the first letter in lowercase and all other letters are uppercase.

Look at the documentation for str for 2 methods you can use to manipulate strings that will give you the result we want.

No looping! That also means no list comprehensions. (If you don't know what that means, don't worry about it).

It should work like this:

```
>>> silly_string = silly_case("This is a string")
>>> silly_string
'tHIS iS a sTRING'
>>> poem = """It matters not how strait the gate,
... How charged with punishments the scroll.
... I am the master of my fate:
... I am the captain of my soul."""
>>> silly_poem = silly_case(poem)
>>> silly_poem
'iT mATTERS nOT hOW sTRAIT tHE gATE,\nhOW cHARGED wITH pUNISHMENTS tHE sCROLL.\ni aM tHE mASTER oF
mY fATE:\ni aM tHE cAPTAIN oF mY sOUL.'
>>> print(silly_poem)
iT mATTERS nOT hOW sTRAIT tHE gATE,
hOW cHARGED wITH pUNISHMENTS tHE sCROLL.
i aM tHE mASTER oF mY fATE:
i aM tHE cAPTAIN oF mY sOUL.
```

This is the last stanza of the poem "Invictus" by the English poet William Ernest Henley (1849–1903)

Function 2: dash_stringify

Edit the `dash_stringify` function so that it takes a *list* of words (strings) and *returns a string* containing all the words from the list with ' - ' in between the words. Note the spaces around the dash.

It should work like this:

```
>>> dash_stringify(['A', 'B', 'C'])
'A - B - C'
>>> fruits = ['Orange', 'Lemon', 'Lime', 'Cherry', 'Peach', 'Apricot']
>>> fruit_string = dash_stringify(fruits)
>>> fruit_string
'Orange - Lemon - Lime - Cherry - Peach - Apricot'
>>> print(fruit_string)
Orange - Lemon - Lime - Cherry - Peach - Apricot
>>> nums = [1, 2, 3]
>>> dash_stringify(nums)
Traceback [...]
 [...]
TypeError: sequence item 0: expected str instance, int found
>>> dash_stringify(['1', '2', '3'])
'1 - 2 - 3'
```

Function 3: is_perfect_square

Edit the function `is_perfect_square` so that it takes a number as input and *returns* `True` if the given number is a perfect square, or `False` if it is not.

A perfect square is an integer number which is the square of an integer. For example 25 (5 * 5), 49 (7 * 7) and 81 (9 * 9) are all perfect squares. In other words, the square root of the number is an integer.

Hint: Look up methods on the "Float" type for something that looks useful. To calculate the square root of a `number`, use `number ** 0.5`.

It should work like this:

```
>>> is_perfect_square(119025) is True
True
>>> is_perfect_square(81) is True
True
>>> is_perfect_square(16)
True
>>> is_perfect_square(25)
True
>>> is_perfect_square(33)
False
>>> is_perfect_square(20)
False
```

Function 4: is_leap_year

Edit the function `is_leap_year` so that it takes a year (number) as input and *returns* `True` if and only if the given year is a leap year. If it is not a leap year, it should return `False`.

- Leap years are years that are divisible by 4
- Exception: centennial years (years divisible by 100) *are not* leap years
- Exception to exception: years divisible by 400 *are* leap years

It should work like this:

```
>>> is_leap_year(2000) is True
True
>>> is_leap_year(2012)
True
>>> is_leap_year(1900)
False
>>> is_leap_year(2018)
False
```

Grading

50 points total: 10 points per function; plus 10 points for things like *following directions* (passing flake8 and turning in the correct file, etc).

FAQ

Q. A student was having an issue with pasting into the REPL, and asked if there is another way to test the functions.

A. Yes! What you can do to test the functions is, instead of pasting the function into the REPL, you can enter:

```
>>> from HW1 import silly_case, dash_stringify, is_perfect_square, is_leap_year
```


Or just import the function you need, and then test it.

```
>>> from HW1 import is_perfect_square
```

We haven't talked about modules yet, which is why I have you paste the functions into the REPL when you are ready to test them. But if you believe me about importing, then it will work. :-)

NOTE: There is one issue with importing the function from the file. Every time you change the code in your file, you have to exit the REPL and restart Python. When something is imported, Python only imports it once, interprets the code and caches it. So if it gets another import command for the same function, it is ignored. This prevents circular dependencies and namespace confusion, but for us it gets very annoying, obviously.

When pasting into the REPL, you can also use the up-arrow to get to the function lines, edit them as necessary to re-define the function.

My email is dianechen.ucsdext@gmail.com. Please do not hesitate to email me if you have questions.

