Homework 4 — Loops, loops, lists

Due Tuesday, October 2nd at 11:55pm

(50 points)

Objectives:

- Understand how iteration and for loops work
- Understand the accumulation pattern and how to implement it using for loops
- Understand how to use string functions and string indexing

Turn In:

- `hw4_loopsloopslists.ipynb`

You should turn this file in on moodle under the "Homework 4 - Loops, Loops, Lists" link.

Instructions:

There is a quiz on moodle that lets you test to make sure that you have correctly named your functions, that they take the correct number of parameters, and that they have the correct return value. It is up to you to fully test your functions. **If your functions are not named exactly as we ask, you will lose points.**

Write a `main()` function. For each of the following functions, call them each at least twice in your main function after your implement them. **Two of these function calls for each function must not be listed as examples in this document for that function to receive full credit.** If your function returns a value, save it in a variable and print it out. **(some of the points for each function go to whether or not you did this in your main function)**

You **must** include the `if __name__ == "__main__"` statement at the bottom of this cell to call your `main()` function!

1) **(10 points)** Write a function, `multiples`, that takes two parameters, say n1 and n2. The function must then print results of n1 * 0, n1 * 1, n1 * 2, …., n1 * n2.

Function call	Output (printed)

`multiples(5, 7)`	`0`
	`5`
	`10`
	`15`
	`20`
	`25`
	`30`
	`35`

`multiples(3, 2)`	`0`
	`3`
	`6`

2) **(10 points)** Write a function, `star_pyramid`, that prints out a star pyramid of the given size (size is a parameter to the function, star_pyramid). Consult the following table for what your pyramid should look like. Note the stars as well as the spaces!

Function call	Output (printed)
`star_pyramid(1)`	`*`
`star_pyramid(2)`	` *` `**`

`star_pyramid(3)`	`  *`
	` **`
	`***`

`star_pyramid(4)`	`   *`
	`  **`
	` ***`
	`****`

3) **(5 points)** Modify your function from question 3rd above, to make the function `star_pyramid_text` so that it takes two parameters, a string and a size and prints out a pyramid made up of the given text for the given size. Note spacing!

Function call	Output (printed)

`star_pyramid_text("cat", 4)`	`         cat`
	`      catcat`
	`   catcatcat`
	`catcatcatcat`

`star_pyramid_text("3", 3)`	`  3`
	` 33`
	`333`

`star_pyramid_text("friend", 1)`	`friend`
`star_pyramid_text("strawberry", 2)`	`          strawberry` `strawberrystrawberry`

4) **(5 points)** Write a function, `factorial`, that computes the factorial of an integer (n!) and **returns** that value. A factorial of a number is computed according to the formula: n! = 1 * 2 * 3 * … * n.

Function call	Return value
`factorial(0)`	`1`
`factorial(2)`	`2`
`factorial(3)`	`6`
`factorial(17)`	`355687428096000`

5) **(5 points)** Write a function, `reverse`, that takes a string as a parameter and returns the reversed version of it.

Function call	Return value
`reverse("bat")`	`"tab"`
`reverse("Apple Pie")`	`"eiP elppA"`

6) **(4 points)** Write a function, `string_stats`, that takes no parameters. It asks the user for a comma-separated list of items, then cycles through them, printing out the number of vowels (a,e,i,o,u) and consonants in each item. The items may contain spaces, which should not be included in either the vowel count or the consonant count. This function should be case-insensitive. You should use the python string `split()` and `count()` functions to solve this problem.

Function call	Output (printed, input in **bold underlined**)

`string_stats()`	`Items?` **`dog,cat,blue-footed booby`**
	`dog`
	`Vowels: 1`
	`Spaces: 0`
	`Consonants: 2`
	
	`cat`
	`Vowels: 1`
	`Spaces: 0`
	`Consonants: 2`
	
	`blue-footed booby`
	`Vowels: 7`
	`Spaces: 1`
	`Consonants: 9`

7) **(3 points)** Write a function, `number_square`, that takes one integer as a parameter and prints out a number square of that size. Watch out for spacing! Hint: you'll want to use a nested for loop here!

`number_square(1)`	`0 0` `0 1`

`number_square(2)`	`0 0 0`
	`0 1 2`
	`0 2 4`

`number_square(3)`	`0 0 0 0`
	`0 1 2 3`
	`0 2 4 6`
	`0 3 6 9`

`number_square(4)`	`0 0 0 0 0`
	`0 1 2 3 4`
	`0 2 4 6 8`
	`0 3 6 9 12`
	`0 4 8 12 16`

8) **(3 points)** Write a function, `absolute_day`, that, given a month and a day, returns the absolute day of the year that day corresponds to. You may assume that leap years do not exist.

Function call	Return value
`absolute_day(1, 1)`	`1`
`absolute_day(3, 24)`	`83`

9) **(bonus, 3 points)** Write a function, `string_square`, that takes one string as a parameter and prints out a string square for that string. Watch out for spacing!

Function call	Output (printed)

`string_square("cat")`	`cat`
	`atc`
	`tca`

`string_square("bird")`	`bird`
	`irdb`
	`rdbi`
	`dbir`

`string_square("ox")`	`ox` `xo`

Comments (5 points)

Your code must be commented. You must include a file comment, inline comments, and function comments. An example function comment is shown below:

```
# This function converts a temperature in fahrenheit to celsius
# and prints the equivalence.
# Parameters: fahrenheit - int or float degrees fahrenheit
# Return: float equivalent temperature in celsius
def fahrenheit_to_celsius(fahrenheit):
      celsius = (fahrenheit - 32) * (5/9)
      return celsius
```