Summer 2016 Bootcamp Problem Set 04

Function Created: countdown(n)

- There is nothing to solve on this first one. We are just going to review a programming technique called **RECURSION**.
- A function that uses recursion solves a problem by repeatedly calling **itself**.
- A recursive function has a **base case** where it stops calling itself and it has a **recurse case** where it repeats a call to itself with slightly altered parameters to continue "solving the problem."
- It is important to note that recursion can be an "expensive" computing operation. So be careful using it. This is just to make you aware of this technique and so that you understand what this is if someone refers to using recursion to solve a problem.
- **Also, any problem that can be solved using recursion can be solved without using recursion! Knowing trivia like this can do wonders for your social life!** ☺

- Type this code in exactly as shown and execute it and see if you can understand it.

```
def countdown(n):
    if n <= 0:
        print 'Takeoff!!'    # base case
    else:
        print n
        countdown(n-1)       # recurse case

countdown(5)
```

- Notice the use of a clear **base case** and a clear **recurse case**. Recursion is very elegant like that. But if you don't set up your base case and recurse case carefully, you can run into errors and crash your computer (if not for Python's runtime helping you)!

- Just to make sure you understand this technique, modify the code you have typed out above to add a line as shown below. **BEFORE** you run it, try and predict what the output will be if the new line you added is uncommented. Run it only **AFTER** you have made a predication! ☺
  When you are ready, uncomment the line that multiplies n by 10 and run it.

```
def countdown(n):
    if n <= 0:
        print 'Takeoff!!'    # base case
    else:
        print n
        countdown(n-1)       # recurse case
        # print n * 10

countdown(5)
```

Function Created: visualize (num)

- Write a **recursive** function that will add the numbers starting from num down to 1. So visualize(5) should give you 5 + 4 + 3 + 2 + 1 = 15.
- Ignore zero and negative numbers and just return 0 for them.

- Remember to use **RECURSION**. Try to think about what the **base case** is and what the **recurse case** is that gets you closer to the base case.

- If this is too new or difficult (for now) of a concept, you can type the sample code below. It will help you visualize what is going on! ☺

```
def visualize (n):
    space = ' ' * (4 * n)
    print space, 'solving', n

    if n <= 0:
        print space, 'BASE returning 0' # base case
        return 0
    else:
        recurse = visualize(n - 1)      # recurse case call
        result = n + recurse
        print space, 'RECURSE returning', result
        return result

print visualize(5)
```

The output of the above function is shown below.

```
                    solving 5
                solving 4
            solving 3
        solving 2
    solving 1
 solving 0
 BASE returning 0
     RECURSE returning 1
         RECURSE returning 3
             RECURSE returning 6
                 RECURSE returning 10
                     RECURSE returning 15
15
```

Summer 2016 Bootcamp Problem Set 04

Function Created: factorial (num)

- You coded **factorial** before.
- Now adapt your code to accept a parameter.
- Try to solve factorial using **RECURSION**! This time**, DON'T** look at the **visualize** sample code and try do this as best as you can from scratch!

- If you like, you can look at the **countdown** example to try and set up your **base case** and **recurse case** for **factorial** using that as a starting point. Here it is.

```
def countdown(n):
    if n <= 0:
        print 'Takeoff!!'    # base case
    else:
        print n
        countdown(n-1)       # recurse case

countdown(5)
```

- So calling factorial (5) should recurse and call factorial(4) and so on until it gets to the base case.
- Your function won't print anything. It will just return the final result of factorial.

Factorials are very simple things. They're just products, indicated by an exclamation mark. For instance, "four factorial" is written as "4!" and means $1 \times 2 \times 3 \times 4 = 24$. In general, n! means the product of all the whole numbers from 1 to n; that is, $n! = 1 \times 2 \times 3 \times \ldots \times n$.

Summer 2016 Bootcamp Problem Set 04

Function Created: **write_file(msg)**

- Prompt the user for a message and write the entered message to a file in the folder you are working in on your computer. You can use a default file name that you make up like output.txt. Also, don't forget the full path to this file on your computer.
- You can google this or find some code on stackoverflow.com. The end of the Code Academy tutorial also made you work with files so you can also review that as well.
- **Note**: Remember to close your file handle. **Also, in your actual coursework, please check with your instructor on using code you find to avoid Academic Dishonesty issues.** For the summer boot camp, we won't worry about that.

Function Created: **read_write (file_to_read, file_to_write)**

- Create a file called in.txt and paste some text into it.
- Read from the file_to_read file (in.txt, for example) and process it line by line writing each line as output to the filename specified in file_to_write (out.txt, for example).
- You can google this or find some code on stackoverflow.com. The end of the Code Academy tutorial also made you work with files so you can also review that as well.
- **Note**: Remember to close your file handles. **Also, in your actual coursework, please check with your instructor on using code you find to avoid Academic Dishonesty issues.** For the summer boot camp, we won't worry about that.

Function Created: **wordcount**

['the', 'cat', 'sat', 'on', 'the', 'wall', 'and', 'the', 'cat', 'sat', 'on', 'the', 'mat',
'where','the','rat','usually','sat','and','the','cat','sat','on','the','rat']

- Calculate and display the word count for each word in this list of words in largest to smallest order.
- Only use the base sequence classes (list, tuple, dictionary). No other specialized classes.

Summer 2016 Bootcamp Problem Set 04

Function Created: indexer

['the', 'cat', 'sat', 'on','the','wall','and','the','cat','sat','on','the','mat',
'where','the','rat','usually','sat','and','the','cat','sat','on','the','rat']

- Display the above list to the user.
- Accept a word as input from the user stopping when they type "stop" case-insensitive.
- Create a list containing the position (or index) of this word.
- Only use the base sequence classes (list, tuple, dictionary). No other specialized classes.
- For example, the word "the" will have a list like this: [1, 5, …]
- Return the resulting list of indices for any given word that is in the list above.

Function Created: cipher

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
'GnWkmEJIRQsHLzPSdqVoxfrZbTyOBXhNiwCDcupUYMKgAjlavtFe'

You are given the two strings above. You have to create a cipher that can encode and decode a given string by the user.

- Prompt the user for a string.
- While the user does not enter 'x' or 'X', continue prompting.
- If the user enters abc, the encoded string would be GnW as shown above.
- If the user enters 1GnW, ignore the 1 and you should calculate and decode the string as abc and print that to the screen. So the 1 at the start of the string is a decode signal versus an encode signal which is the absence of 1 at the start of the string. **Note**: Other than using the 1 as a signal, we don't decode the 1.
- Don't use any specialized classes. Just a loop and raw_input.

Summer 2016 Bootcamp Problem Set 04

Function Created: analyze (data_file, results_file)

- Create one or more **data_files** (called 001.txt and 002.txt below) and paste some comma-separated numbers into it as shown. The filename and the comma-separated values you use are up to you.
- The **analyze** function will read in a line at a time from a given data_file. It will tokenize the line values and add the values to a list.
- It will then pass the list to your previously coded basic_stats function from Problem Set 03 (or use my solution, if you like) and write the resulting stats into the **results_file** as shown below.
- There will be one line of output for one line of input.
- **data_file** and **results_file** will contain the path and name to each respective file on your computer.

001.txt 9,8,7,6 7,3,2,1,43,5,2	**001_stats.txt** [6, 7, 8, 9] ---> {'min': 6, 'max': 9, 'median': 7.5, 'range': 3, 'total': 30, 'mean': 7.5} [1, 2, 2, 3, 5, 7, 43] ---> {'min': 1, 'max': 43, 'median': 3.0, 'range': 42, 'total': 63, 'mean': 9.0}
002.txt 5,4,2,7,5,3,2,3,5 1,10	**002_stats.txt** [2, 2, 3, 3, 4, 5, 5, 5, 7] ---> {'min': 2, 'max': 7, 'median': 4.0, 'range': 5, 'total': 36, 'mean': 4.0} [1, 10] ---> {'min': 1, 'max': 10, 'median': 5.5, 'range': 9, 'total': 11, 'mean': 5.5}

- You can google this or find some code on stackoverflow.com. The end of the Code Academy tutorial also made you work with files so you can also review that as well.
- **Note**: Remember to close your file handles. **Also, in your actual coursework, please check with your instructor on using code you find to avoid Academic Dishonesty issues.** For the summer boot camp, we won't work about that.

Summer 2016 Bootcamp Problem Set 04

Function Created: hangman

- Maintain a list of words, for e.g. ['abnormal', 'horse', 'kitty']
- Randomly choose a word from the list as the word to guess. Let us say the word is `'kitty'`, you would display to the user:

  `_ _ _ _ _`

- Give the user (word length + 3) chances to guess letters. So the user would have 8 chances for this word.
- Accept a guess as input from the user.
- If the guess is a letter in the word, "flip" the dash. For example if the user guesses 't', you would display:

  `_ _ t t _`

- If they guess a letter that does not exist in the word, tell them the letter does not exist and print the current status of the word. For e.g., if the user's next guess is 'p', then you would notify them it does not exist and print the current status of the word

- Continue until the user runs out of guesses or solves the word, whichever occurs first. Notify them accordingly.

Function Created: palindrome

- Palindromes are spelled the same way forwards and backwards.
- Accept a string as input from the user.
- Indicate if it is a palindrome or not.
- Don't use the [::-1] method.
- Try to iterate over the string forwards and backwards without using the previous approach and make a decision that way.

Summer 2016 Bootcamp Problem Set 04

Function Created: odometer

- Print all the possible odometer values that match the conditions specified in the paragraph below.

"I was driving on the highway the other day and I happened to notice my odometer. Like most odometers, it shows six digits, in whole miles only. So, if my car had 300,000 miles, for example, I'd see 3-0-0-0-0-0."

"Now, what I saw that day was very interesting. I noticed that the last 4 digits were palindromic; that is, they read the same forward as backward. For example, 5-4-4-5 is a palindrome, so my odometer could have read 3-1-5-4-4-5."

"One mile later, the last 5 numbers were palindromic. For example, it could have read 3-6-5-4-5-6. One mile after that, the middle 4 out of 6 numbers were palindromic. And, are you ready for this? One mile later, all 6 were palindromic!"

"The question is, what was on the odometer when I first looked?"

More Practice

- Remember to use Python Coding Bat or Learning Python the Hard Way if you would like more practice with coding.
- I will post the solutions and another problem set in a week after everyone has had a chance to attempt these problems.
- If you have other recommendations for coding practice resources, feel free to post to our Piazza site.
- Have fun! ☺