CMPS 1500 Introduction to Computer Science I – Spring 2019

Lab 7

Problem 0 is due on Zybook at the end of the lab session.
All other problem are due **Wednesday 4/3/19** at 11:59 p.m. on Canvas and Zybook

Please follow the usual lab guidelines (file naming rules, honor code, comment requirements) and review the following lab rules.

- The code will be graded both on functionality and on style (meaningful variable names, presence of comments and docstrings, absence of commented-out code, etc). Perfectly running programs will lose points for lack of comments, presence of commented-out code, etc
- Your code has to run. Any code that immediately crashes will receive 0 points.
- Whenever you submit code on Canvas, please submit the entire code (function and main part), so that we can run the code.

Read the entire assignment before starting your work in order to plan your time.

0. **Together in the lab, `lab7pr0.py`, Zybook**

   Modified lab rules for this problem only: Work on the problems in this exercise **together** in the lab. You can team up with one or more students without having to list their names, and you can discuss hints and (partial) solutions together with the instructor in the lab. This is your time to practice, so make the best use of it.

   Upload the debugged function `factorial_product(n)` and the function `allCaps(s)`, including the helper function `caps(s)` to Zybook. You do not have to upload the mergesort function trace.

   (a) **Mergesort function tracing**
   Draw a function trace for `mergesort([5,25,55,1])`.
   This is important practice for problem 2(c) below.

   (b) **Debugging exercise**
   Consider the following code:

```
def factorial_product(n):
    if n<=1:
        return ""
    else:
        return str(n)+"*"+factorial_product(n)

print(factorial_product(5))
```

   The function `factorial_product(n)` is supposed to return a string representation of the product of numbers to compute `n`, e.g., `factorial_product(5)` should return the string `"5*4*3*2*1"` . But the function is not correct as written. Your task is to debug this function.

i. Run the code in IDLE. What kind of error do you get when you try to run it? (The name of the error is shown in the last line of the error message.) Why do you get this error, and how can you fix it?

ii. To fix the remaining errors, it might be helpful to add a print statement inside the function to see what the value of `n` is in each call.

(c) **Recursive function warmup**

The goal of this exercise is to write a recursive function `allCaps(s)` that has a string parameter `s` and returns a new string that has all letters capitalized. Ay non-letter characters are not changed. For example, `allCaps("Roll Wave!")` returns `ROLL WAVE!`.

i. Start first by developing a helper function `caps(c)` to capitalize the single character `c`, i.e., `caps("a")` returns `"A"`, and `caps("Z")` returns `"Z"`. This function is not recursive.

ii. Now develop a recursive function `allCaps(s)` that uses the `caps(c)` function. What is the recursive case? What is the base case?

1. **Recursive functions, `lab7pr1.py`, Zybook**

(a) Write a **recursive** function `uppercount(s)` that counts the number of uppercase letters in the input string.

Do not use loops or built-in Python string functions. The only string functions you can use are slicing and indexing.

```
>>>> n = uppercount("Hello, World")
print(n)
2
```

Add comments to your code to point out the base case and the recursive case.

(b) Write a **recursive** function `clean_string(s)` that returns a new "clean" string in which all symbols that are not uppercase or lowercase letters or spaces have been removed from `s`.

Do not use loops.

```
>>>> CS = clean_string("#Hello, world42")
print(CS)
Hello world
```

Add comments to your code to point out the base case and the recursive case.

(c) Using a similar idea to the one you employed for the `clean_string(s)` function, write a recursive function `clean_list(l1, l2)` that takes two lists as input and returns a list of elements from `l1` that are not present in l2.

Do not use loops.

```
>>>> unique = clean_list([1,2,3,4,5,6,7], [2,4,6])
print(unique)
[1,3,5,7]
```

Add comments to your code to point out the base case and the recursive case.

2. **Recursion vs. iteration,** `lab7pr2.pdf`**, Canvas**
   For this question turn in a single report document (`lab7pr2.pdf`) that contains all answers to all the subproblems. I.e., it should contain the code of the function `F(n)` and `f(n)`, the function trace, and the runtime evaluation.

   As we discussed, recursion is a very powerful problem-solving technique, any computation can be expressed recursively. However, recursion doesn't always lead to efficient solution, as you will now see.

   A classic example of recursion is the definition of Fibonacci number. Wikipedia has a good article about it: `http://en.wikipedia.org/wiki/Fibonacci_number` (Remember to check the part that talks about Fibonacci numbers in nature). The sequence of Fibonacci numbers is recursively defined as follows: $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$ for $n > 1$. So, the first few Fibonacci numbers are $0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

   (a) Write a recursive function `F(n)` that returns the n-th Fibonacci number. Add comments to your code to point out the base case and the recursive case.

   (b) Write an iterative function (using iteration, i.e., loop(s)) `f(n)` that returns the n-th Fibonacci number.
   Hint: If you just look at the sequence $0, 1, 1, 2, 3, 5, 8, 13, 21$*, how do you compute the next Fibonacci number? It's 13+21, so 44. Use this rule repeatedly in a loop.*

   (c) For the recursive version `F(n)`, draw a function trace for $F(5)$ that shows all the recursive function calls and their return values.

   Please scan your trace and add it to the report document (or you can draw one electronically). The function trace should illustrate what function calls what other function(s) and with what parameters, as discussed in class and in the lab session.

   (d) Just like you did in lab 6, measure the time it takes to compute the 10th, 20th, 30th and 40th Fibonacci numbers using your iterative and recursive functions. Copy the values into the report document, where you make a table illustrating n and the time it took to reach the solutions. In a couple of sentences compare the efficiency of the recursive and iterative solutions, and explain why it takes long to compute larger Fibonacci numbers.

3. **Cybersecurity,** `lab7pr3.txt`**, Canvas**
   In the past weeks you explored applications of computing in different fields of study. In the next few weeks you'll be looking at the impact computing technology is making on societies and humanity in general.

   Watch the TED talk by James Lyne "Everyday cybercrime - and what you can do about it" (17 minutes):

   `http://www.ted.com/talks/james_lyne_everyday_cybercrime_and_what_you_can_do_about_it`

   (a) One of the types of crime that computing technology made extremely easy to achieve is identity and personal data theft. In your opinion, why is identity theft a problem? How can identifying information be misused? What about

other types of personal data? (Think about 2-3 different areas of one's life, e.g. what if your grade information is stolen? What about your health records? Or purchase history?) Explain your point in 3-6 sentences.

(b) In the talk, the speaker mentions several ways of making oneself vulnerable by exposing too much data. For instance, posting photos taken at home publicly allows the entire world to see your home address (GPS coordinates) if geotagging is not turned off. Which of these steps are true about you (and which are not)? After watching this talk, do you believe it is possible to completely avoid exposing personal data? Why or why not? Provide a 3-6 sentence answer.