Algorithms for Applications | HW0: *Python Basics*
CS 5112 Fall 2023 | **Due: September 19, 11:59pm ET**

Homework Instructions: The homework contains mainly three types of tasks:

- **Coding.** You can find a template for all the coding problems inside the `problems.zip` in the module. Please use these templates, to ensure the files and functions have the same names that the autograder will check for.
- **Algorithm Description.** When a questions requires designing an algorithm, you should describe what your algorithm does in the writeup, in clear concise prose. You can cite algorithms covered in class to help your description. You should also argue for your algorithm's asymptotic run time or, in some cases when indicated in the problem, for the run time bounds of your implementation.
- **Empirical Performance Analysis.** Some questions may ask you to do an empirical performance analysis of one or more algorithms' runtime under different values of n. For these questions, you should generate at least ten test cases for the implementation, for various values of n (include both small and large instances). Then, measure the performance locally on your own system, by taking the median runtime of the implementation over ten or more iterations. Graph the resulting median run times. The x-axis should be instance size and the y-axis should be median run time. You can use any plotting library of your choice. If you have never plotted on Python before, this matplotlib turorial has some examples.

  If several subquestions in a problem ask you to provide performance analysis, please do them all in the same environment (same system and configuration), in order to have a more accurrate comparison between algorithms. You can include graphs for a same problem in the same plot, as long as the different graphs are properly labeled.

  For these types of questions, you will not need to submit the tests you generate or your code for benchmarking or plotting the algorithms. However, you should include the graphs you generate into the write-up. You should also write about what you observe from the analysis and potentially compare it to the complexity analysis. Finally, also include a description of the environment (CPU, operating system and version, amount of memory) you did the testing on.

Submission Instructions: Hand in your solutions electronically on Gradescope. There are two active assignments for this problem set on Gradescope: one with the autograder, where you submit your code, and another one where you submit the write-up.

Coding Submission: You only need to submit the code for the specific functions we ask you to implement. While you will write additional code to generate the performance analysis graphs when required, you do not need to submit that code.

To submit to the autograder, please zip the `problems` directory we provided you with. Therefore, your zipped file should have the following structure:

```
problems
└── problem_1
```

```
│
│   ├── p1_a.py
│   ├── p1_b.py
│   └── ...
├── problem_2
│   └── p2.py
└── ...
```

In order to ensure your code runs correctly, do not change the names of the files or functions we have provide you. Additionally, do not import external python libraries in these files, you don't need any libraries to solve the coding assignments. Finally, if you write any helper functions for your solutions, please include them in the same files as the functions that call them.

We have some public tests in the autograder to verify that all the required files are included, so you will be able to verify on Gradescope that your work is properly formatted shortly after submitting. We strongly encourage checking well ahead of the due date that your solutions work on the autograder, and seeking out assistance from the TAs should it not. We cannot guarantee being responsive the night the assignment is due.

Write-up Submission: Your write-up should be a nicely formatted PDF prepared using the LaTeX template on Canvas, where you can type in your answer in the `main.tex` file. If you do not have previous experience using LaTeX, we recommend using Overleaf. It is an online LaTeX editor, where you can upload and edit the template we provide. For additional advice on typesetting your work, please refer to the *resources* directory on the course's website.

Academic Integrity: You may use online sources or tools (such as code generation tools), but any tools you use should be explicitly acknowledged and you must explain how you used them. You are responsible for the correctness of submitted solutions, and we expect you to understand them and be able to explain them when asked by teaching staff.

Collaboration Policy: Collaboration (in groups of up to three students) is encouraged while solving the problems, but:

1. list the netids of those in your group;
2. you may discuss ideas and approaches, but you should not write a detailed argument or code outline together;
3. notes of your discussions should be limited to drawings and a few keywords; you must **write up the solutions and write code on your own**.

In this mini-assignment, you are expected to gain a basic understanding of basic pythonic concepts, as well as basic recursion and dynamic programming. This assignment is generated largely with chatGPT, to cover the basics of Python.

Students are expected to solve these problems by browsing the internet to find relevant documentation to enable them to solve the problems. **We strongly advise against** using chatGPT to solve these basic problems.

Problem 1. p1_a:

In this problem set, we warm-up on:

(a) If-Elif-Else statements

    1. categorize_number

(b) Enumerating lists, reversing lists.

    1. find_index
    2. reverse_list

(c) For loops with break and continue statements

    1. process_numbers

(d) Building a basic dictionary

    1. string_lengths

(e) Conditional list comprehensions

    1. even_values_keys
    2. square_evens
    3. long_strings
    4. convert_to_celsius

Problem 2. p1_b:

In this problem set, we warm-up on lists. Specifically:

(a) Appending

    1. add_element

(b) Inserting by index

    1. insert_element_at

(c) Removing by index

    1. remove_element

(d) Popping from list

    1. pop_element_at

Problem 3. p1_c:

In this problem set, we warm-up on dictionaries. Specifically:

(a) Keys, values, items, get

    1. get_keys
    2. get_values
    3. get_items
    4. get_value

Problem 4. p1_d:

In this problem set, we warm-up on:

(a) Basic list comprehension

    1. square_all
    2. filter_even

(b) reduce and lambda functions

    1. product_of_all

Problem 5. p1_e:

In this problem set, we warm-up on:

(a) Applying reduce-lambda for calculating factorials. Provide a basic write-up describing your solution for this problem.

    1. factorial_reduce

(b) Applying recursion for calculating factorial. Provide a basic write-up describing your solution for this problem.

    1. factorial_rec

Problem 6. p1_advanced:

In this problem set, we give four advanced problems which warm-up or provide a gentle introduction to:

(a) Appending, popping, removing and inserting to lists.

    1. manage_queue

(b) Applying dynamic programming for calculating factorial

    1. factorial_dp

(c) Simple linear search problem

    1. linear_search

(d) Binary search problem. Provide a basic write-up describing your solution for this problem, as well as provide a graph comparing the performance of linear_search and binary_search for list of the following sizes: [10, 100, 1000, 10000, 100000, 1000000].

    1. binary_search