CS37300 Homework — Homework 0— Last update: Jan 14, 2024
Due: Jan 24, 2024 (11:59 pm)

Instructions and Policy: You are allowed to study with others and use online resources for reference, however, the work you turn in must be your own. This means do not copy/paste from Stack Exchange (or from another student.) If you have worked closely with other students, provide their name(s) and a brief (at most one paragraph) description of the interaction; if we feel this oversteps the bounds, we will discuss it with you. Each student should write up their own solutions independently.

The requirements below are supposed to be followed in this and further homework assignments.

- For your theoretical submission:
  - YOU MUST INCLUDE YOUR NAME IN THE HOMEWORK.
  - The answers MUST be submitted via Gradescope.
  - Please write clearly and concisely - clarity and brevity will be rewarded. Refer to known facts as necessary.
  - Theoretical questions MUST include the intermediate steps to the final answer.
- For your programming answers:
  - The Python scripts will be submitted separately via Gradescope
  - Zero points will be given in any question where the Python code answer doesn't match the answer on Gradescope.
  - If the answer is/includes a plot, it should be added to your theoretical submission unless otherwise specified.

Your code is REQUIRED to run on Python 3.11 in an environment detailed in Homework 0 under the Python Installation section. While your code may run in other environments, any points lost due to environmental issues will not be regraded.

Please make sure you don't use any libraries that are not imported for you. If such a library is used, you will get 0 pt for the coding part of the assignment.

If your code doesn't run on Gradescope, then even if it compiles on another computer, it will still be considered not running and the respective part of the assignment will receive 0 pt.

Python Installation

If you already have a Python distribution installed, you can still use it for this class. However, you will need to install some packages using `pip` or `conda`, and it is your responsibility to make sure your code runs in the environment specified below. If you are not familiar with `pip` or `conda`, you should install Anaconda.

In this class, you need a Python 3.11 distribution. The best way to get it is to install Anaconda (`https://www.anaconda.com/products/individual`), which is a Python distribution that comes with a lot of useful packages for scientific computing. It also comes with a package manager called `conda` that makes it easy to install, uninstall, and update packages.

After installation, you should be able to run Python from the terminal. Open a terminal and type `python`. You should see something like this:

```
Python 3.11.x | packaged by Anaconda, Inc. | (xxxxxx) [xxxxxx] on xxxxx
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Then, run `conda -V` to check the version of `conda`. You should see something like this:

```
conda xx.x.x
```

Now, create a virtual environment that is specifically used for this class. A virtual environment is a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages. It allows you to work on a specific project without worrying about affecting other projects. To create a virtual environment, run the following command. Enter `y` when prompted to proceed.

This command will only need to be run once.

```
conda create -n cs373 python=3.11
```

When the installation is done, you can activate the virtual environment by running the following command whenever you want to work on this class.

```
conda activate cs373
```

For each assignment, you will be given a `environment.yml` file that specifies the packages you need to install. To install the packages, run the following command.

```
conda env update --file environment.yml
```

To use the virtual environment in VSCode (highly recommended) or any other IDE, you need to select the Python interpreter in the virtual environment. In VSCode, you can do this by clicking the Python version in the bottom right corner and selecting the Python interpreter in the virtual environment.

Programming Part (11+9+13=33 pts)

Overview

For this assignment, please download the provided `hw0_handout.zip` file from **Brightspace**.

In this assignment, you will learn and use some basic Python and Numpy skills. You will also learn how to load and visualize data using Pandas and Matplotlib. The goal of this assignment is to get you familiar with the tools we will use in this class.

Files

You will fill in functions in `python_basics.py`, `numpy_basics.py`, and `data_basics.py`. After you finish, you will submit these three files as well as any other files mentioned in the assignment to Gradescope.

Asides from these three files, you will also find `utils.py` and `exam_original.csv` in the same folder. You do not need to modify these two files. `utils.py` contains some helper functions, and `exam_original.csv` is the data you will use in this assignment.

Packages

You will need the following packages for this assignment:

- `numpy`
- `pandas`
- `matplotlib`

These packages should be installed if you have installed Anaconda. If you are not using Anaconda, you can install them using pip or conda. For example, you can run `pip install numpy` in the terminal to install `numpy`.

Evaluation

Unless otherwise specified, your code will be evaluated by an autograder on Gradescope using the same environment as detailed above. You should make sure your code runs without errors in that environment. Otherwise, you may lose points. You can submit your code to Gradescope as many times as you want. We will only grade the latest submission.

There will be two types of test cases in the autograder: public (local) and hidden. Public test cases are visible to you, and you can see the results after you submit your code. You can also run the public test cases locally by running `python <filename>.py` in the same folder as your code. Passing all public test cases **does not** guarantee you will get full credit for the assignment.

Hidden test cases, however, are not visible to you, and you will not be able to see the results until your grade is published. The final score you get for this assignment is the sum of the scores of all public and hidden test cases.

Getting Help

If you have any questions about this assignment, please contact the course staff for help, preferably during office hours. You can also post your questions on the course forum. However, please do not post any code publicly. When asking questions, you should describe your problem in words and post the relevant code snippet. Please avoid showing a screenshot of your code to TAs and ask "why my code is not working". This will not help you get an answer faster.

1 Python Basics (11 pts)

Getting Started

To avoid unexpected errors, you should open the folder `hw0_handout` in VSCode. You can do this by clicking `File` → `Open Folder` and selecting the folder `hw0_handout`. You should see the following files in the Explorer panel on the left side of the VSCode window.

Alternatively, on Windows, you can right-click the folder `hw0_handout` and select `Open with Code`. On MacOS, you can drag the folder `hw0_handout` to the VSCode icon in the dock.

This is to make sure that the Python source files are not in a nested folder, which could cause problems when generating figures.

Please note that using Jupyter Notebook or similar tools is highly discouraged due to their interactive nature which does not define some variables in the global scope. However, if you choose to use them, you will need to make sure that your code runs in the environment specified above.

Understanding Python Type Hints and DocTests

While you may have some Python backgrounds, you may not have seen the following syntax before. This is called type hints and is used to specify the types of the input and output of a function. Type hints are not required nor enforced by Python but are meant to help you understand the code better. For example, the function signature below means that the function `mystery_function` takes two integers a and b and a tuple of two integers as input and returns an integer.

```python
def mystery_function(a: int, b: int, c: Tuple[int, int]) -> int:
 """
 This is a docstring. It is used to describe the function.
 Below are some examples of doctests.
 >>> mystery_function(1, 2, (3, 4))
 10
 >>> mystery_function(1, 2, (3, 4, 5)) # This works but does not follow the type hints.
 10
 >>> mystery_function(1, 2) # This would cause an error.
 Traceback (most recent call last):
 ...
 TypeError: mystery_function() missing 1 required positional argument: 'c'
 """
 return a + b + c[0] + c[1]
```

You should also notice that we have added some tests to the docstrings of the functions. These tests are called doctests. The code following `>>>` is the input to a Python interpreter, and the code after that is the expected output. The doctests in the skeleton code are served as examples and also local tests, and we have programmed the skeleton code such that you can run the local tests by running python `python_basics.py` in the same folder as your code.

Since doctests assert exact output match character by character, so in rare cases, local tests may produce false positives because of some specific implementation details or environment settings. If you feel that your code is correct but the local tests fail, try submitting your code to Gradescope. If Gradescope is happy with your code, you should be fine.

1.1 (2 pts) OOP in Python - Constructor

In `class Multiplication`, fill in the constructor (`__init__`) so that it takes two numbers first and second as input and stores them as attributes `self.first` and `self.second`. You do not need to return anything. Notice the keyword `self`, which is a reference to the current instance of the class and is similar to `this` in Java.

1.2 (1 pts) OOP in Python - Method

In `class Multiplication`, fill in the method `multiply` so that it calculates the product of `self.first` and `self.second`, stores the product as an attribute `self.answer`, and returns the product.

1.3 (2 pts) Print and Format

In `class Multiplication`, fill in the method `display` so that it prints the answer in the following format:

```
First: 1.00
Second: 2.00
Product: 2.0000
```

Note that both values for First and Second should be printed with two decimal places and the value for Product should be printed with four decimal places. There are many ways to do this. You can use `print` and `format` or `f-string`. You can also use `round` to round the answer to four decimal places.

1.4 (2 pts) Data Structures - List

In `class DataStructure`, fill in the method `generate_list` so that it returns a list containing 10 random digits between 0 and 4 (inclusive). You should use `random.randint` to generate random integers. Do not include random.seed() in your code!

1.5 (2 pts) Data Structures - Set

In `class DataStructure`, fill in the method `find_unique` so that it returns a set containing the unique elements in the input list `my_list`. You can use `set` to create a set from a list.

1.6 (2 pts) Data Structures - Dictionary

In `class DataStructure`, fill in the method `count_frequency` so that it returns a dictionary containing the frequency of each element in the input list `my_list`. You do not need to sort the dictionary by any order, as we have done that for you in the return statement.

2 Numpy Basics

In this part, you will look through the Numpy Quickstart Tutorial and learn how to use Numpy. You will fill in the functions in `numpy_basics.py`. You can run the autograder locally by running `python numpy_basics.py` in the same folder as your code

2.1 (3 pts) Numpy Basics - Array

In `class NumpyBasics`, fill in the method `split_list` so that it splits the input list `x` into two arrays `first_half` and `second_half` and returns them. The length of `x` is not necessarily even, so you should make sure that the length of `first_half` is always greater than or equal to the length of `second_half`. **DO NOT use any loops**.

Note that you should use Numpy arrays instead of Python lists. You can use `np.array` to create a Numpy array from a Python list when needed.

2.2 (3 pts) Numpy Basics - Functions

In `class NumpyBasics`, fill in the method `multiply_inverse` so that it takes a Numpy array `x` and a Numpy array `y` as input and returns the inverse of the product of `x` and `y`. You may assume such an operation is always valid. **DO NOT use any loops**.

2.3 (3 pts) Numpy Basics - Broadcasting

In `class NumpyBasics`, fill in the method `custom_normalization` that applies the following normalization to the input Numpy array `x` and returns the normalized array:

- Replace all negative values with the minimum value in `x`.
- Multiply each value in the new array by the value at the same position in `x`.
- Return an array with each value being the square root of the value at the same position after the previous step.

DO NOT use any loops.

3 Playing around with Data

In this part, you will learn how to load and visualize data using Pandas and Matplotlib. You will fill in the functions in `data_basics.py`. You can run the autograder locally by running `python data_basics.py` in the same folder as your code.

If you are using PyCharm, you may fail some of the test cases because the default terminal width in PyCharm might be too narrow. You can fix this by opening the "Run/Debug Configurations" window and clicking "Edit configurations ->Python ->Modify options" and checking "Emulate terminal in console output."

3.1 (1 pt) Loading Data

In `class DataBasics`, fill in the constructor (`__init__`) so that it loads the data from the CSV file named `filename` into a Pandas DataFrame and stores it as an attribute `self.df`. You can use `pd.read_csv` to load the data.

3.2 (3 pts) Preprocessing Data

In `class DataBasics`, fill in the method `preprocess` so that it preprocesses the data in `self.df` and returns the preprocessed DataFrame. Since this is an introductory assignment, the preprocessing steps are simple:

- Drop the column `Unnamed: 0`.
- Rename the column `ParentEduc` to `ParentEducation`

After preprocessing, print the first 3 rows of the DataFrame.

3.3 (2 pts) Exploring Data

In `class DataBasics`, fill in the method `reading_stats_given_score` so that it returns the mean and standard deviation of the reading score for students who scored at least `score` in reading.

3.4 (3 pts) Visualizing Data - Histogram

In `class DataBasics`, fill in the method `generate_reading_score_histogram` so that it plots a histogram of the reading score. The histogram should have 20 bins and include appropriate labels and title. The graph is automatically saved as `reading_score_histogram.png` in the same folder as your code after you run the autograder. Don't forget to upload the graph to Gradescope, where the TAs will grade your graph manually.

3.5 (4 pts) Visualizing Data - Scatter Plot

In `class DataBasics`, fill in the method `generate_reading_writing_scatterplot` so that it plots a scatter plot of the reading score and writing score (writing score in y-axis vs reading score in x-axis). The scatter plot should have appropriate labels and a title. You also need to set the edge color of the points to white and transparency to 0.9 so that the points are easier to see. After that, add a regression line in the scatter plot. You may find `np.polyfit` useful.

The graph is automatically saved as `math_writing_scatterplot` in the same folder as your code after you run the autograder. Don't forget to upload the graph to Gradescope, where the TAs will grade your graph manually.

Submission

You will submit the following files to Gradescope. Do not put your source files in a nested folder.

- `python_basics.py`

- `numpy_basics.py`
- `data_basics.py`

You will also submit the following graphs to Gradescope:

- `reading_score_histogram.png`
- `math_writing_scatterplot.png`