CS 7638 Robotics: AI Techniques - Problem Set 0

Fall 2020 - Due Monday, August 24, Midnight AOE

Introduction

This problem set is designed to help you setup Python environment in your local and provide a refresher on concepts that will be useful in the problem sets and projects. If you are totally new to Python, you can refer to https://docs.python.org/3/tutorial/index.html to understand the basics of Python. All the problem sets and projects in this course are designed to use Python 3, and we recommend Python 3.8.

Download PS0 files

On Canvas, click on `Files` in the left side menu, select `Problem Sets`, click on `PS0` and download the files into a directory/folder, named `PS0`. For this problem set, you will be required to make changes to the file `PS0.py` and test it locally by running the file `PS0_Tests.py`.

Conda setup

1. Conda: Conda is an open source package and environment management system. We recommend using Conda since it makes it easy to install and manage different versions of libraries without messing up with other environments. Install Miniconda https://docs.conda.io/en/latest/miniconda.html. It doesn't matter whether you use 2.7 or 3.8 because we will create our own environment anyways.
2. On Windows, open the installed "Conda prompt" to run this command. On MacOS and Linux, you can just use a terminal window. Change directory (using `cd`) to the location of the directory `PS0` (the directory that contains the files `PS0.py`, `PS0_Tests.py`, `rait_env.yml`, `ProblemSet0.pdf` you may have downloaded from Canvas).
3. Create a conda environment, by running the following command in "Conda Prompt" (Windows) or Terminal (MacOS/Linux):

```
conda env create -f rait_env.yml
```

4. This should create an environment named 'rait_env'. Activate it using the following Windows command: `activate rait_env` or the following MacOS / Linux command: `source activate rait_env`

PyCharm Setup

You may choose to use any Python IDE including PyCharm, Visual Studio Code, Sublime, etc, or you may also use just a plain editor and a command line. We show below the steps to setup PyCharm with conda environment that we created above.

Please note that the instructions below are for high-level guidance specific to Linux for PyCharm Professional 2020.2. The exact paths or options may differ for you based on your system. You may refer to the provided PyCharm links in the steps if your operating system or PyCharm version is different.

1. Download and setup PyCharm https://www.jetbrains.com/pycharm/download.
2. Open the directory `PS0` in PyCharm https://www.jetbrains.com/help/pycharm/opening-reopening-and-closing-projects.html#opening_projects.

3. Configure PyCharm to use the conda environment created above https://www.jetbrains.com/help/pycharm/conda-support-creating-conda-virtual-environment.html.

   a: Press Ctrl+Alt+S to open the project Settings/Preferences.

   b. In the Settings/Preferences dialog, select `Project <project name> | Python Interpreter`. Click the icon next to the Python Interpreter dropdown and select Add.

   c. In the left-hand pane of the Add Python Interpreter dialog, select Conda Environment.

   d. Select Existing Environment.

   e. Click Select an interpreter and specify a path to the Conda executable in your file system. To see the path of the conda environment in your system, run the command `conda info --envs` and note the path of the `rait_env`. In the Interpreter path on PyCharm, add that path to the rait_env environment, followed by `bin/python`. An example path would be `/home/user/anaconda3/envs/rait_env/bin/python`.

   f. Select the checkbox Make available to all projects

   g. Apply the changes.

4. Now, to run the file PS0_Tests.py, right click on it in the Project window in the left and select Run.

Windows users may refer to `https://www.youtube.com/watch?v=1gtHso20YMQ` for installing Miniconda and PyCharm if you face issues.

MacOS users may refer to `https://www.youtube.com/watch?v=yQo1kb0_8EI` for installing Miniconda and PyCharm if you face issues.

Quizzes

All the quizzes use Python classes. Please refer to https://docs.python.org/3/tutorial/classes.html for a refresher on classes in Python if required. You are required to make changes to only PS0.py, and you can test if your solution is working by running the file PS0_Tests.py.

Quiz 1

Consider a bank account which provides three operations:

1. `deposit` to add an amount parametrized by a non-negative integer `amount`.
2. `withdraw` to withdraw an amount parametrized by a non-negative integer `amount`. You can be assured that `amount` will never be greater than the balance at any time, so you need not worry about that scenario.
3. `getBalance` to fetch the balance at any given time.

The bank account starts with a zero balance, so you can initialize the balance as zero in the `__init__` method.

You need to edit the class Quiz1 in file PS0.py for this quiz. Gradescope (our autograder) will invoke the above three methods repetitively in some fashion to ensure that the implementation is correct. As an example, the below sequence of lines should run without an error.

```
account1 = PS0.Quiz1()
assert account1.getBalance() == 0
account1.deposit(100)
account1.deposit(20)
```

```
assert account1.getBalance() == 120
account1.withdraw(50)
assert account1.getBalance() == 70
account1.deposit(20)
assert account1.getBalance() == 90
```

The file PS0_Tests.py runs such tests on PS0.py.

Quiz 2

This time we are maintaining the number of dimes and quarters instead of the balance. There are the following operations available:

1. `__init__`: initialization of the number of dimes and quarters.
2. `addCoins`: to add dimes and quarters as specified in the parameter dictionary `dimes_and_quarters`.
3. `removeCoins`: to remove dimes and quarters as specified in the parameter dictionary `dimes_and_quarters`.
4. `getCoins`: returns the number of dimes and quarters we have after additions and removals in a dictionary.
5. `getBalanceCents`: returns the balance from the remaining dimes and quarters.

Note that unlike quiz 1, here we may or may not start with zero dimes and/or quarters, so pay attention to the parameter in the `__init__` method. Please refer to the docstring of the respective methods of class `Quiz2` in PS0.py for more details.

An example of sequence of the methods that should run without an error is:

```
quiz2 = PS0.Quiz2({'dimes': 5})
quiz2.addCoins({'dimes': 2, 'quarters': 10})
assert quiz2.getBalanceCents() == 320
assert quiz2.getCoins() == {'dimes': 7, 'quarters': 10}
quiz2.removeCoins({'dimes': 2, 'quarters': 10})
assert quiz2.getCoins() == {'dimes': 5, 'quarters': 0}
assert quiz2.getBalanceCents() == 50
```

Quiz 3

This quiz is almost same as quiz 2, except that the input parameter type for `addCoins` and `removeCoins` methods is a tuple instead of dictionary. There are the following operations available:

1. `__init__`: initialization of the number of dimes and quarters.
2. `addCoins`: to add dimes and quarters as specified in the parameter tuple `dimes_and_quarters`.
3. `removeCoins`: to remove dimes and quarters as specified in the parameter tuple `dimes_and_quarters`.
4. `getBalanceCents`: returns the balance from the remaining dimes and quarters.

Please refer to the docstring of the respective methods of class `Quiz3` in PS0.py for more details. An example sequence of the methods that should run without an error is:

```
quiz3 = PS0.Quiz3({'dimes': 5})
quiz3.addCoins((2, 10))
assert quiz3.getBalanceCents() == 320
quiz3.removeCoins((2, 10))
assert quiz3.getBalanceCents() == 50
```

Quiz 4

It is common to make mistakes while coding, and a traceback of the error is the most helpful thing in debugging errors. This quiz already has all the code implemented, but there is an error when we execute it. The task in this quiz is to get rid of the error and get back the intended value.

There are two methods that this quiz provides: 1. `__init__` which initializes the accounts of different customers. 2. `displayBalance` which returns a string to display the balance for all the customers.

An example of a sequence of lines that will be executed in Gradescope and that should run without error is:

```
quiz4 = PS0.Quiz4({ 'Alex': (5, 10), 'Bob': (0, 2) })
assert quiz4.displayBalance() == '(Customer: Alex, Balance: 300)(Customer: Bob, Balance: 50)'
```

Testing

We have provided a file `PS0_Tests.py` that has a few sample test cases. Make the required changes to `PS0.py` and you can run this file in your local to test. The final score is out of 100.

Submission

Make the required changes to the file `PS0.py` and submit only this file to Gradescope.