# CS 7638 Robotics:AI Techniques - Problem Set 0

Spring 2024 - Due Monday, May 20, 2024, 11:59 PM - AOE

[Note that PS0 does not count towards your grade.]

### Introduction

This is a problem set to introduce some basic Python features, ensure your environment is correctly set up, and practice submitting to Gradescope. It is **ungraded**, but you should still complete it.

## Working with Python

All the questions have you write methods in 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 `ps0_answers.py`, and you can test if your solution is working correctly by running the same locally.

You will need to have set up your Python interpreter and class environment to complete this problem set, so refer to the Environment Setup pdf for guidance.

## Checking your answers

In order to check your answers on any of the problem set files you will need to have the `checkutil.py` file located in the same directory. Simply execute: `python ps0_answers.py` to see your results.‘

### Question 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. `get_balance` 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 Question1 in file ps0_answers.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 = Question1()
assert account1.get_balance() == 0
account1.deposit(100)
account1.deposit(20)
assert account1.get_balance() == 120
account1.withdraw(50)
assert account1.get_balance() == 70
account1.deposit(20)
assert account1.get_balance() == 90
```

The file ps0_answers.py includes test cases that can be run from the command line.

### Question 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. `add_coins`: to add dimes and quarters as specified in the parameter dictionary `dimes_and_quarters`.
3. `remove_coins`: to remove dimes and quarters as specified in the parameter dictionary `dimes_and_quarters`.
4. `get_coins`: returns the number of dimes and quarters we have after additions and removals in a dictionary.
5. `get_balance_cents`: 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 `Question2` in ps0_answers.py for more details.

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

```
account2 = Question2({'dimes': 5})
account2.add_coins({'dimes': 2, 'quarters': 10})
assert account2.get_balance_cents() == 320
assert account2.get_coins() == {'dimes': 7, 'quarters': 10}
account2.remove_coins({'dimes': 2, 'quarters': 10})
assert account2.get_coins() == {'dimes': 5, 'quarters': 0}
assert account2.get_balance_cents() == 50
```

## Question 3

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

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

Please refer to the docstring of the respective methods of class `Question3` in ps0_answers.py for more details. An example sequence of the methods that should run without an error is:

```
account3 = Question3({'dimes': 5})
account3.add_coins((2, 10))
assert account3.get_balance_cents() == 320
account3.remove_coins((2, 10))
assert account3.get_balance_cents() == 50
```

## Question 4

It is common to make mistakes while coding, and a traceback of the error is the most helpful thing in debugging errors. This question 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. `display_balance` 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:

```
account4 = Question4({ 'Alex': (5, 10), 'Bob': (0, 2) })
assert account4.display_balance() == '(Customer: Alex, Balance: 300)(Customer: Bob, Balance: 50)'
```

## Testing

There are several test cases provided. Make the required changes to `ps0_answers.py` and you can run this file locally to test before submitting. The final score is out of 100.

## Submission

Make the required changes to the file `ps0_answers.py` and submit to Gradescope once complete.