Homework 0 --- Programming

Content adapted from Berkeley CS188

Introduction

The projects for this class assume you use Python 3.10.

Project 0 will cover the following:

- Instructions on how to set up the right Python version,
- Workflow examples,
- A mini-Python tutorial,
- Project grading: Every project's release includes its autograder for you to run yourself.

Files to Edit and Submit: You will fill in portions of `addition.py`, `buyLotsOfFruit.py`, and `shopSmart.py` in tutorial.zip .

Evaluation: Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, section, and the discussion forum are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Discussion: Please be careful not to post spoilers.

Python Installation

Many of you will not have Python 3.10 already installed on your computers. Conda is an easy way to manage many different environments, each with its own Python versions and dependencies. This allows us to avoid conflicts between our preferred Python version and that of other classes. We'll walk through how to set up and use a conda environment.

On Windows, to execute these commands, you need to be in an Anaconda prompt.

Prerequisite: Anaconda.

Creating a Conda Environment

The command for creating a conda environment with Python 3.10 is:

```
conda create --name <env-name> python=3.10
```

For us, we decide to name our environment `cs471`, so we run the following command, and press `y` to confirm installing any missing packages.

```
[cs471-ta ~/python_basics]$ conda create --name cs471 python=3.10
```

Entering the Environment

To enter the conda environment that we just created, do the following. Note that the Python version within the environment is 3.10, just what we want. Note: if you have an old Anaconda installation, you will need to use source instead of conda for the below, or for Windows, not type conda at all.

```
[cs471-ta ~/python_basics]$ conda activate cs471
(cs471) [cs471-ta ~/python_basics]$ python -V
```

```
Python 3.10.16
```

Note: the tag (`<env-name>`) shows you the name of the conda environment that is active. In our case, we have (`cs471`), as what we'd expect.

Leaving the Environment

Leaving the environment is just as easy.

```
(cs471) [cs471-ta ~/python_basics]$ conda deactivate
[cs471-ta ~/python_basics]$ python -V
Python 3.9.10 :: Anaconda custom (x86_64)
```

Our python version has now returned to whatever the system default is!

Workflow/ Setup Examples

You are not expected to use a particular code editor or anything, but here are some suggestions on convenient workflows (you can skim both for half a minute and choose the one that looks better to you):

- GUI and IDE, with VS Code shortcuts. You are highly encouraged to read the Using an IDE section if using an IDE to learn convenient features.
- In terminal, using Unix commans and Emacs (this is fine to do on Windows too). Useful to be able to edit code on any machine without setup, and remote connecting setups such as using the instructional machines.

Python Basics

If you're new to Python or need a refresher, we recommend going through the Python basics tutorial.

Autograding

To get you familiarized with the autograder, we will ask you to code and test the autograder after solving the three questions.

You can obtain all of the files associated the autograder tutorial in the tutorial.zip . Unzip this file and examine its contents:

```
[cs471-ta ~]$ unzip tutorial.zip
[cs471-ta ~]$ cd tutorial
[cs471-ta ~/tutorial]$ ls
addition.py
autograder.py
buyLotsOfFruit.py
grading.py
projectParams.py
shop.py
shopSmart.py
testClasses.py
testParser.py
test_cases
tutorialTestClasses.py
```

This contains a number of files you'll edit or run:

- `addition.py`: source file for question 1
- `buyLotsOfFruit.py`: source file for question 2
- `shop.py`: source file for question 3
- `shopSmart.py`: source file for question 3
- `autograder.py`: autograding script (see below)

and others you can ignore:

- `test_cases`: directory contains the test cases for each question
- `grading.py`: autograder code
- `testClasses.py`: autograder code
- `tutorialTestClasses.py`: test classes for this particular project
- `projectParams.py`: project parameters

The command `python autograder.py` grades your solution to all three problems. If we run it before editing any files we get a page or two of output:

> Click to see full output of `python autograder.py`

For each of the three questions, this shows the results of that question's tests, the questions grade, and a final summary at the end. Because you haven't yet solved the questions, all the tests fail. As you solve each question you may find some tests pass while other fail. When all tests pass for a question, you get full marks.

Looking at the results for question 1, you can see that it has failed three tests with the error message "add(a, b) must return the sum of a and b". The answer your code gives is always 0, but the correct answer is different. We'll fix that in the next tab.

Q1: Addition

Open `addition.py` and look at the definition of `add`:

```
def add(a, b):
    "Return the sum of a and b"
    "*** YOUR CODE HERE ***"
    return 0
```

The tests called this with a and b set to different values, but the code always returned zero. Modify this definition to read:

```
def add(a, b):
    "Return the sum of a and b"
    print("Passed a = %s and b = %s, returning a + b = %s" % (a, b, a + b))
    return a + b
```

Now rerun the autograder (omitting the results for questions 2 and 3):

```
[cs471-ta ~/tutorial]$ python autograder.py -q q1
Starting on 1-22 at 23:12:08

Question q1
===========

*** PASS: test_cases/q1/addition1.test
***     add(a,b) returns the sum of a and b
*** PASS: test_cases/q1/addition2.test
***     add(a,b) returns the sum of a and b
*** PASS: test_cases/q1/addition3.test
***     add(a,b) returns the sum of a and b

### Question q1: 1/1 ###


Finished at 23:12:08

Provisional grades
==================
Question q1: 1/1
------------------
Total: 1/1
```

You now pass all tests, getting full marks for question 1. Notice the new lines "Passed a=..." which appear before "*** PASS: ...". These are produced by the print statement in `add`. You can use print statements like that to output information useful for debugging.

Q2: buyLotsOfFruit function

Implement the `buyLotsOfFruit(orderList)` function in `buyLotsOfFruit.py` which takes a list of `(fruit,numPounds)` tuples and returns the cost of your list. If there is some `fruit` in the list which doesn't appear in `fruitPrices` it should print an error message and return `None`. Please do not change the `fruitPrices` variable.

Run `python autograder.py` until question 2 passes all tests and you get full marks. Each test will confirm that `buyLotsOfFruit(orderList)` returns the correct answer given various possible inputs. For example, `test_cases/q2/food_price1.test` tests whether:

```
Cost of [('apples', 2.0), ('pears', 3.0), ('limes', 4.0)] is 12.25
```

Q3: shopSmart function

Fill in the function `shopSmart(orderList,fruitShops)` in `shopSmart.py`, which takes an orderList (like the kind passed in to `FruitShop.getPriceOfOrder`) and a list of `FruitShop` and returns the `FruitShop` where your order costs the least amount in total. Don't change the file name or variable names, please. Note that we will provide the `shop.py` implementation as a "support" file, so you don't need to submit yours.

Run `python autograder.py` until question 3 passes all tests and you get full marks. Each test will confirm that `shopSmart(orderList,fruitShops)` returns the correct answer given various possible inputs. For example, with the following variable definitions:

```
orders1 = [('apples', 1.0), ('oranges', 3.0)]
orders2 = [('apples', 3.0)]
dir1 = {'apples': 2.0, 'oranges': 1.0}
shop1 =  shop.FruitShop('shop1',dir1)
dir2 = {'apples': 1.0, 'oranges': 5.0}
shop2 = shop.FruitShop('shop2', dir2)
shops = [shop1, shop2]
```

`test_cases/q3/select_shop1.test` tests whether: `shopSmart.shopSmart(orders1, shops) == shop1`

`and test_cases/q3/select_shop2.test` tests whether: `shopSmart.shopSmart(orders2, shops) == shop2`

Submission

Nothing to do here.