Lab 0 Solutions **lab00.zip (lab00.zip)**

Solution Files

Introduction

This lab explains how to use your own computer to complete assignments for CS 61A and introduces some of the basics of Python.

If you need any help at any time through the lab, please feel free to come to office hours (/office-hours) or post on Piazza (https://piazza.com/berkeley/summer2022/cs61a).

The setup in this lab is required to complete all other assignments in the course, so this lab is *required*, and you may not use a lab drop on this assignment. If you joined the course late, you should still complete this lab, and you can request an extension (/articles/about#assignment-extensions) so you can still turn it in for points.

To complete the assignment, you must complete all of the steps from the Doing the assignment (/lab/lab00/#doing-the-assignment) section onwards, including the WWPD question, the code-writing question, and submitting with OK.

This lab looks really long, but it's mostly setup and learning how to use the essential tools for this class; these may seem a bit difficult now, but will quickly become second nature as we move further into the course.

Here's a breakdown of the major parts of the lab:

- **Setup**: Setting up the essential software you'll use to turn in lab, homework, and project assignments for the course. This will require several components, listed below. If you already have these components set up on your computer (e.g. you already have a terminal installed, or you already have Python 3.9 installed) you can skip this portion of the lab.
    - **Install a terminal**: Install a terminal so you can interract with files in this course and run OK commands—if you have a terminal on your computer and feel comfortable using it, you can skip this part.
    - **Install Python 3**: Install the Python programming langauge to your computer—if you already have Python installed, you can skip this part, but you must have Python 3.7 or later (ideally Python 3.9).
    - **Install a text editor**: Install software to edit `.py` files for this course (e.g. VSCode, Atom, etc.)—you can skip this part if you already have a text editor you like.
    - **Backup setups**: These are options you can use if you have difficulty getting any of the above components to work on your personal computer— you should ask

for help from your TA as well!

- **Walkthrough: Using the terminal**: This walks you through how to use the terminal on your computer, as well as the Python interpreter. If you already feel comfortable with both of these you do not need to read this section.
- **Walkthrough: Organizing your files**: This section walks you through how to use your terminal to organize and navigate files for this course. **Everyone should at least skim this section**, as it has important information specific to this class, but if you are already comfortable navigating directory structures with a terminal much of this will feel familar.
- **Review: Python basics**: This is a review on many of the basic components of Python introduced in lecture. You should have already seen this material, but we like to include a brief review of relevant content on each lab in case you need a refresher on anything.
- **Required: Doing the assignment**: You must complete this section to get points for the assignment. Here you will practice the different types of problems you will be asked to do in lab, homework, and project assignments for this course. The problems themselves will (hopefully) not seem too difficult—the main goal of this assignment is to give you practice using our software.
- **Required: Submitting the assignment**: You must complete this section to get points for the assignment. This will walk you through how to turn in your work after completing the previous section, and how to verify that your work is turned in on OKPY.
- **Appendix: Useful Python command line options**: These are commands that are useful in debugging your work, but not required to complete the lab. We include them because we imagine they're likely to be helpful to you throughout the course.

Required: Doing the assignment

> When working on assignments, ensure that your terminal's working directory is correct (which is likely where you unzipped the assignment).

What Would Python Do? (WWPD)

One component of lab assignments is to predict how the Python interpreter will behave.

Enter the following in your terminal to begin this section:

```
python3 ok -q python-basics -u
```

You will be prompted to enter the output of various statements/expressions. You must enter them correctly to move on, but there is no penalty for incorrect answers.

The first time you run Ok, you will be prompted for your bCourses email. Please follow these directions (/articles/using-ok/#signing-in-with-ok). We use this information to associate your code with you when grading.

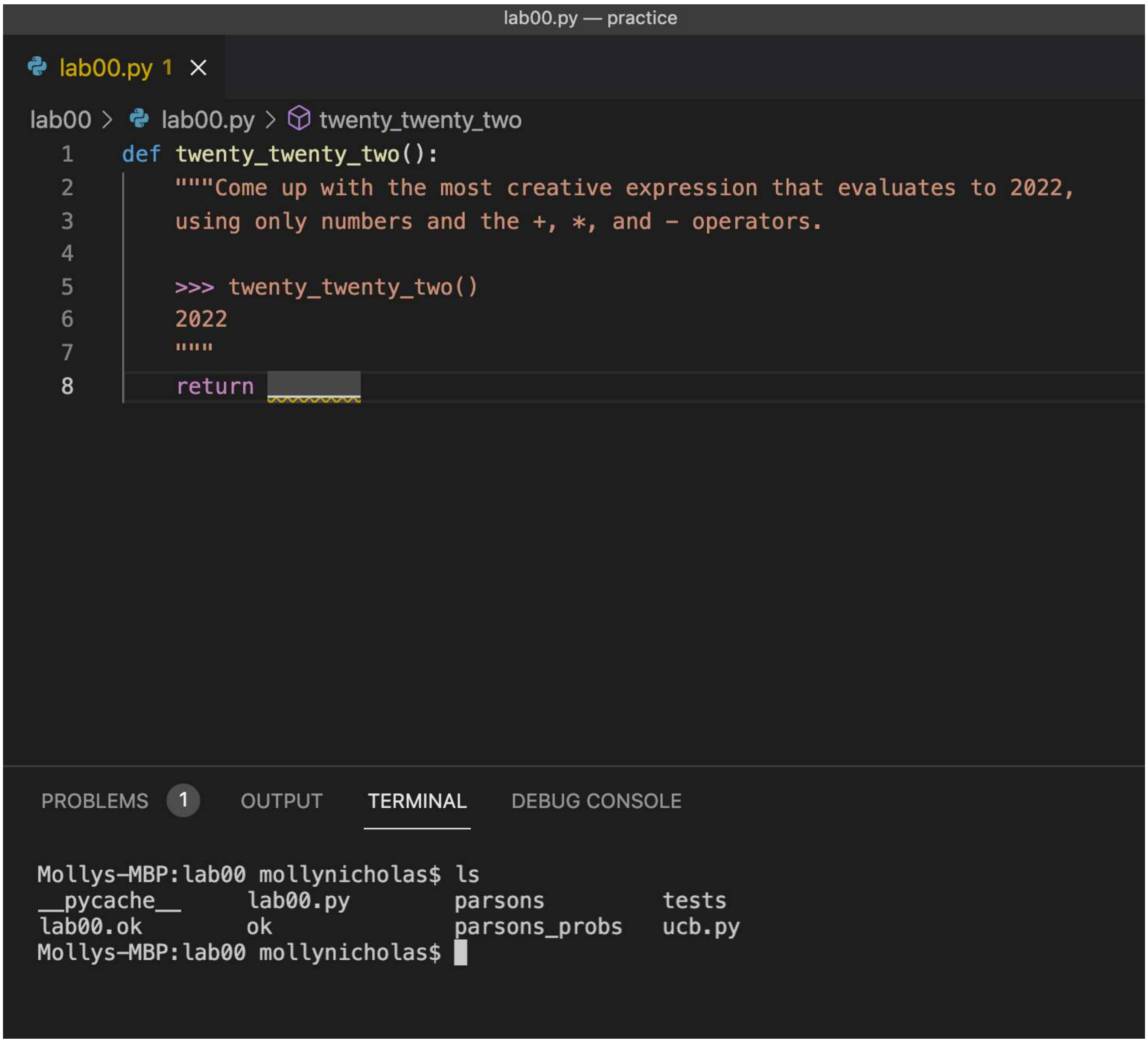
```
>>> 10 + 2
______

>>> 7 / 2
______

>>> 7 // 2
______

>>> 7 % 2  # 7 modulo 2, the remainder when dividing 7 by 2.
______
```

```
>>> x = 20
>>> x + 2
______

>>> x
______

>>> y = 5
>>> y = y + 3
>>> y * 2
______

>>> y = y // 4
>>> y + x
______
```

Code writing questions

Understanding problems

Labs will also consist of function writing problems. Open up `lab00.py` in your text editor. You can type `open .` on MacOS or `start .` on Windows to open the current directory in your Finder/File Explorer. Then double click or right click to open the file in your text editor. You should see something like this:



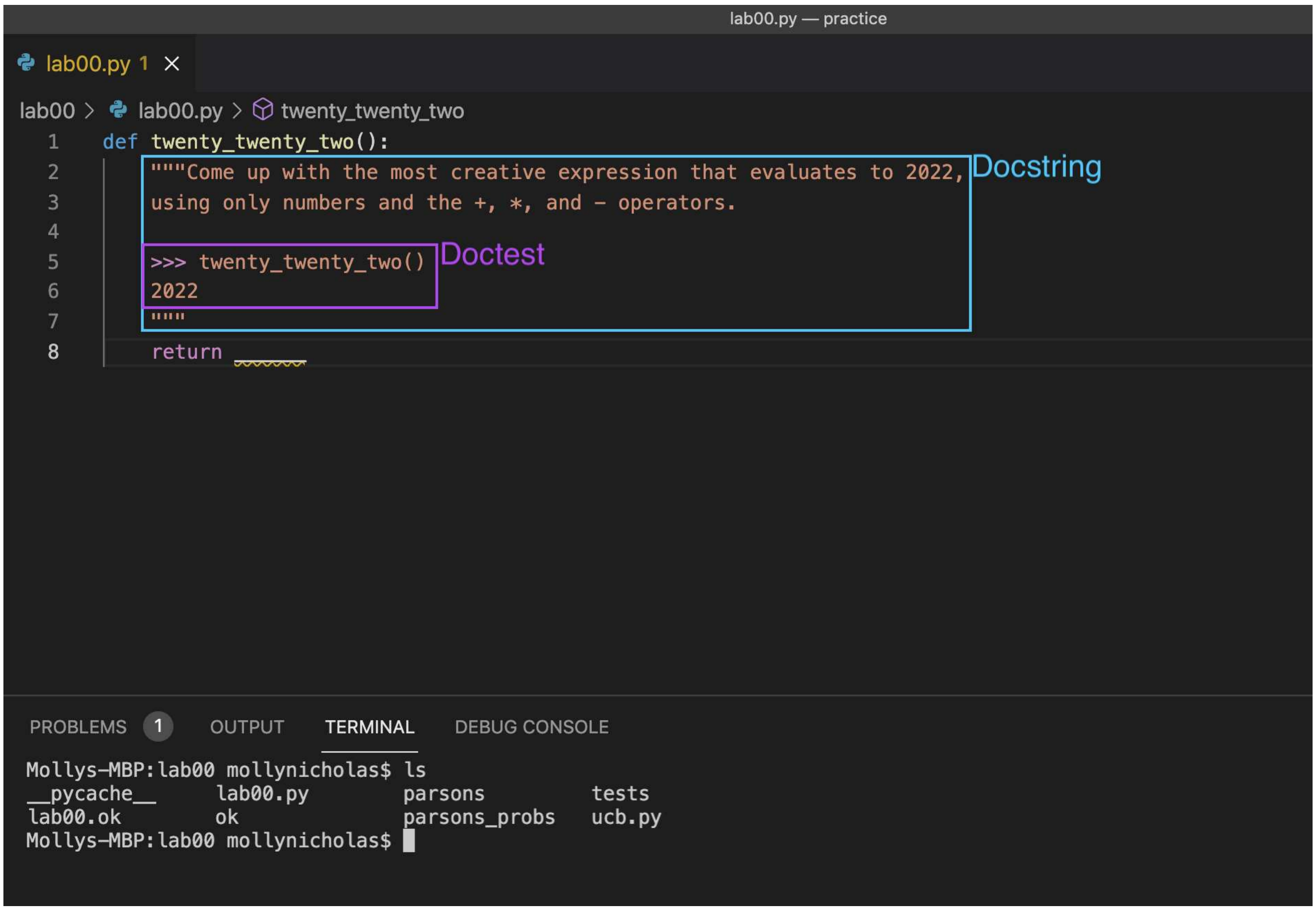
```python
def twenty_twenty_two():
    """Come up with the most creative expression that evaluates to 2022,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty_two()
    2022
    """
    return ______
```

```
Mollys-MBP:lab00 mollynicholas$ ls
__pycache__     lab00.py        parsons         tests
lab00.ok        ok              parsons_probs   ucb.py
Mollys-MBP:lab00 mollynicholas$ 
```

The lines in the triple-quotes `"""` are called a **docstring**, which is a description of what the function is supposed to do. When writing code in 61A, you should always read the docstring!

The lines that begin with `>>>` are called **doctests**. Recall that when using the Python interpreter, you write Python expressions next to `>>>` and the output is printed below that line. Doctests explain what the function does by showing actual Python code. It answers the question: "If we input this Python code, what should the expected output be?"

Here, we've circled the docstrings and the doctests to make them easier to see:



```python
def twenty_twenty_two():
    """Come up with the most creative expression that evaluates to 2022,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty_two()
    2022
    """
    return ______
```

```
Mollys-MBP:lab00 mollynicholas$ ls
__pycache__     lab00.py        parsons         tests
lab00.ok        ok              parsons_probs   ucb.py
Mollys-MBP:lab00 mollynicholas$ 
```

In `twenty_twenty_two` ,

- The docstring tells you to "come up with the most creative expression that evaluates to 2022," but that you can only use numbers and arithmetic operators `+` (add), `*` (multiply), and `-` (subtract).
- The doctest checks that the function call `twenty_twenty_two()` should return the number 2022.

> You should not modify the docstring, unless you want to add your own tests! The only part of your assignments that you'll need to edit is the code unless otherwise specified.

Writing code

Once you understand what the question is asking, you're ready to start writing code! You should replace the underscores in `return ______` with an expression that evaluates to 2022. What's the most creative expression you can come up with?

> Don't forget to save your assignment after you edit it! In most text editors, you can save by navigating to File > Save or by pressing Command-S on MacOS or Ctrl-S on Windows.

Running tests

In CS 61A, we will use a program called `ok` to test our code. `ok` will be included in every assignment in this class.

> For quickly generating ok commands, you can now use the ok command generator (https://go.cs61a.org/ok-help).

Back to the terminal—make sure you are in the `lab00` directory we created earlier (remember, the `cd` command lets you change directories).

In that directory, you can type `ls` to verify that there are the following three files:

- `lab00.py` : the starter file you just edited
- `ok` : our testing program
- `lab00.ok` : a configuration file for Ok

Now, let's test our code to make sure it works. You can run `ok` with this command:

```
python3 ok
```

> Remember, if you are using Windows and the `python3` command doesn't work, try using just `python` or `py` . See the the install Python 3 section for more info and ask for help if you get stuck!

If you wrote your code correctly and you finished unlocking your tests, you should see a successful test:

```
=====================================================================
Assignment: Lab 0
Ok, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    3 test cases passed! No cases failed.
```

If you didn't pass the tests, `ok` will instead show you something like this:

```
---------------------------------------------------------------------
Doctests for twenty_twenty_two

>>> from lab00 import *
>>> twenty_twenty_two()
2013

# Error: expected
#     2022
# but got
#     2013

---------------------------------------------------------------------
Test summary
    0 test cases passed before encountering first failed test case
```

Fix your code in your text editor until the test passes.

> Every time you run Ok, Ok will try to back up your work. Don't worry if it says that the "Connection timed out." We won't use your backups for grading.
>
> While `ok` is the primary assignment "autograder" in CS 61A, you may find it useful at times to write some of your own tests in the form of doctests. Then, you can try them out using the `-m doctest` option for Python).
>
> Your progress in the Parsons web app should also be reflected on your terminal using the `python3 ok` commands. You can confirm this by checking on `ok`: `python3 ok -q ilove61a`.

Required: Submitting the assignment

Now that you have completed your first CS 61A assignment, it's time to turn it in. You can follow these next steps to submit your work and get points.

Step 1: Submit with `ok`

In your terminal, make sure you are in the directory that contains `ok`. If you aren't there yet, you can use this command:

```
cd ~/Desktop/cs61a/lab/lab00
```

Next, use `ok` with the `--submit` option:

```
python3 ok --submit
```

This will prompt you for an email address if you haven't run Ok before. Please follow these directions (/articles/using-ok/#signing-in-with-ok), and refer to the troubleshooting steps on that page if you encounter issues. After that, Ok will print out a message like the following:

```
Submitting... 100% complete
Submission successful for user: ...
URL: https://okpy.org/...
```

Step 2: Verify your submission

You can follow the link that Ok printed out to see your final submission, or you can go to okpy.org (https://okpy.org). You will be able to view your submission after you log in.

> Make sure you log in with the same email you provided when running `ok` from your terminal!

You should see a successful submission for Lab 0.

Congratulations, you just submitted your first CS 61A assignment!

> More information on Ok is available here (/articles/using-ok/). You can also use the `--help` flag:
>
> ```
> python3 ok --help
> ```
>
> This flag works just like it does for UNIX commands we used earlier.

Appendix: Useful Python command line options

When running a Python file, you can use options on the command line to inspect your code further. Here are a few that will come in handy. If you want to learn more about other Python command-line options, take a look at the documentation (https://docs.python.org/3.9/using/cmdline.html).

- Using no command-line options will run the code in the file you provide and return you to the command line. For example, if we want to run `lab00.py` this way, we would write in the terminal:

  ```
  python3 lab00.py
  ```

- **`-i`**: The `-i` option runs your Python script, then opens an interactive session. In an interactive session, you run Python code line by line and get immediate feedback instead of running an entire file all at once. To exit, type `exit()` into the interpreter

prompt. You can also use the keyboard shortcut `Ctrl-D` on Linux/Mac machines or `Ctrl-Z Enter` on Windows.
If you edit the Python file while running it interactively, you will need to exit and restart the interpreter in order for those changes to take effect.
Here's how we can run `lab00.py` interactively:

```
python3 -i lab00.py
```

- **`-m doctest`**: Runs doctests in a particular file. Doctests are surrounded by triple quotes (`"""`) within functions.
  Each test in the file consists of `>>>` followed by some Python code and the expected output (though the `>>>` are not seen in the output of the doctest command).
  To run doctests for `lab00.py`, we can run:

```
 python3 -m doctest lab00.py
```