

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, as well as introduce some of the basics of Python. If you are using a lab computer, most of the instructions are the same, except you won't have to install anything.

If you need any help at any time through the lab, please feel free to come to office hours (<https://cs61a.org/office-hours.html>) or post on piazza.

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

Setup

Register for an account

These accounts allow you to use instructional machines in the CS department, which can be useful if you do not have regular access to a computer. They are **not required if you do not plan on using the lab computers or printers.**

Install a terminal

The terminal is a program that allows you to interact with your computer by entering commands. No matter what operating system you use (Windows, macOS, Linux), the terminal will be an essential tool for CS 61A.

macOS/Linux

If you're on a Mac or are using a form of Linux (such as Ubuntu), you already have a program called Terminal or something similar on your computer. Open that up and you should be good to go.

Windows

For Windows users, we recommend a terminal called Git-Bash. You can try either method below:

- **Easy (automatic) method:** Right-click here, click *Save link as...* (assets/Install-Python-on-Windows.jse), then save and run our automated installer, following the displayed instructions.

(*Advanced users:* If double-clicking doesn't work, you can try `cscript "Install-Python-on-Windows.jse"` from the Command Prompt.)

If this method **succeeds**, you can move on and install a text editor; you now run Python inside Git-Bash. (In fact, *only* inside Git-Bash. So this *won't* let you run `.py` files by double-clicking them. But you shouldn't need to do that for this course.)

If this method **fails**, try the alternate method below.

Anti-virus note: Some anti-viruses mistakenly flag our installer as a virus. We're aware of this, but there's little we can do to prevent it. In the case of Windows Defender, some students have reported success by clicking "See Details", letting it scan the file, and then rebooting. Otherwise, you can try to exclude/whitelist the file from scanning. If neither of these work for you or you aren't comfortable messing with your anti-virus software, you can just use the manual installation method below.

- **Alternate (manual) method:**

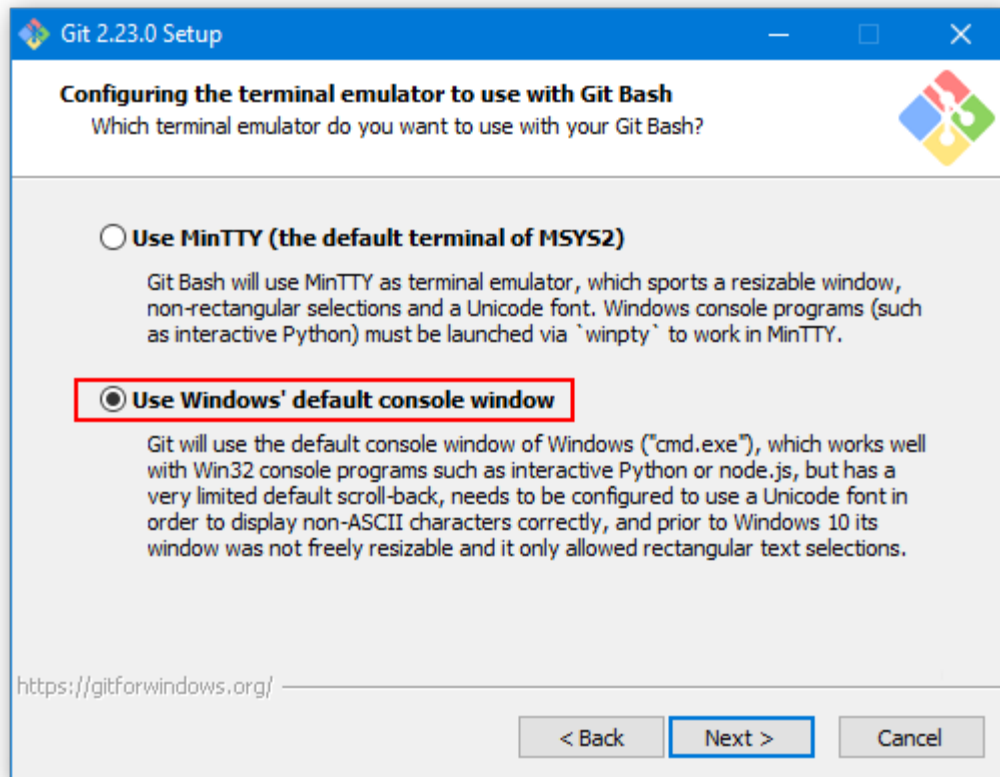
First, if you already tried the automatic installer above, make sure it's fully cleaned up:

1. Look for *Git* as an installed program in "Add/Remove Programs", and, if it exists, uninstall it.
2. Once Git is no longer installed, if a `C:\Program Files\Git` folder still exists, delete that too.

Now download and install Git Bash (<https://git-scm.com/download/win>). You can use the default options, **with one exception**:

Select ***Use Windows' default console window*** in the *Configuring the terminal emulator to use with Git Bash* step.

This is very important! If you do not select this option, your terminal won't work with Python!



If you're already using Git-Bash from outside this course and reinstalling it isn't an option:

Depending on whether you selected the MinTTY option when installing Git, it's possible that typing a command like `python` won't display anything on the screen. You can fix this by typing `winpty python` instead (or `winpty python3`, etc. as described below), but it will be painful, as you will have to remember to do this *every time* for the rest of the course! Hence we recommend that you go back and reinstall Git-Bash with the recommended options if at all possible.

If everything succeeded, you are now able to launch a terminal on Windows by running Git-Bash.

SSL/TLS errors: If you ran into connection security errors, you may need to update your system and/or enable TLS 1.2 (e.g. see here (<https://support.microsoft.com/en-us/help/3140245>) for Windows 7). You can check your TLS version by installing Python first, and running the following in python3 :

```
from json import loads; from urllib.request import urlopen;
loads(urlopen('https://www.howsmyssl.com/a/check').read().decode('UTF-8'))
```

If you don't see TLS 1.2 or later, that may be why you are encountering problems.

If you're on Windows 10, though, the problem may be something else, and you may need to search/ask for help.

Install Python 3

Python 3 is the primary programming language used in this course. Use the instructions below to install the Python 3 interpreter.

(The instructions may feature older versions of Python 3, but the steps are similar.)

Linux

Run `sudo apt install python3` (Ubuntu), `sudo pacman -S python3` (Arch), or the command for your distro.

macOS

Download and install Python: for Windows click here (<https://www.python.org/ftp/python/3.8.1/python-3.8.1-amd64.exe>), for Mac or Linux click here (<https://www.python.org/downloads/>).

Refer to this video (<https://www.youtube.com/watch?v=smHuBHxJdK8>) for additional help on setting up Python.

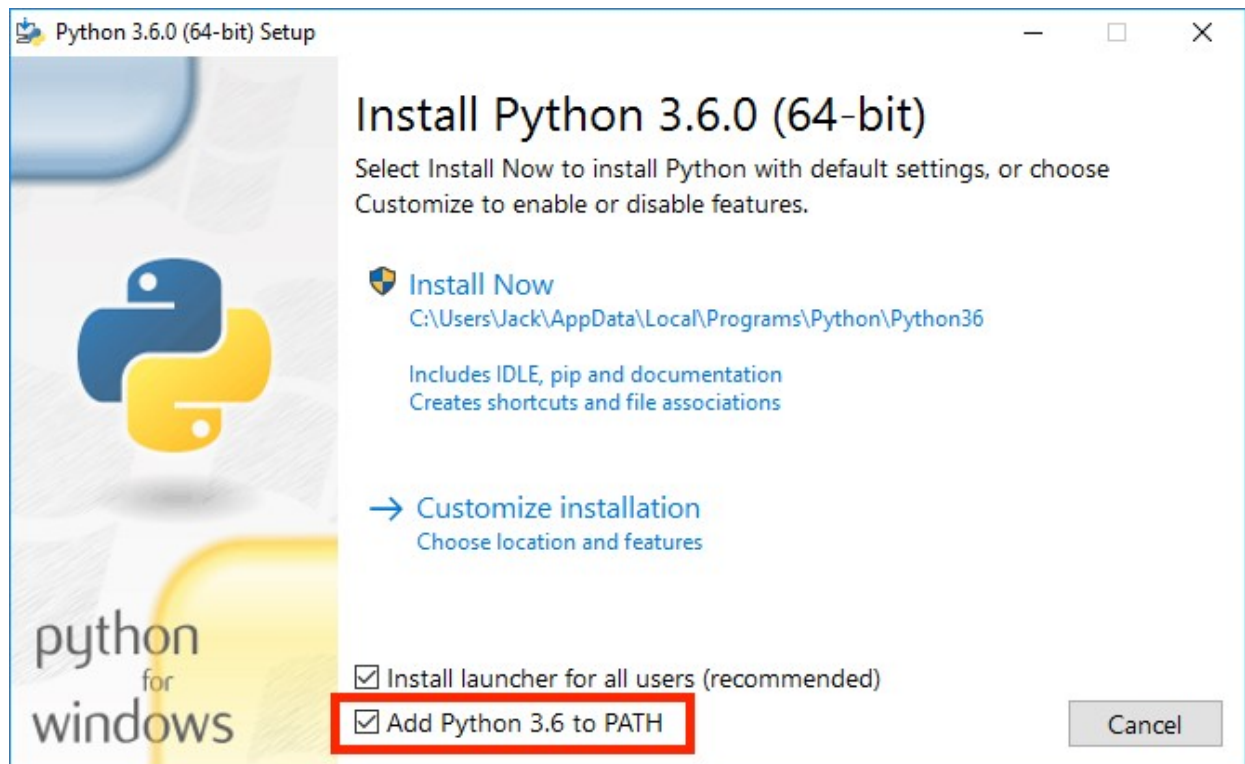
You may need to right-click the download icon and select "Open". After installing please close and open your Terminal.

Windows

If you used our automated installer successfully, skip to the next section—you should already have Python.

Otherwise, if you're installing manually, download Python () and **make sure to check the "Add Python 3.x to PATH" box**, which will allow you to execute the `python` command from your terminal.

After installing please close and open your Terminal.



Other

Download Python from the download page ().

Install a text editor

The **Python interpreter** that you just installed allows you to *run* Python code. You will also need a **text editor**, where you will *write* Python code.

There are many editors out there, each with its own set of features. We find that Atom (<https://atom.io/>) and Sublime Text 3 (<http://www.sublimetext.com/3>) are popular choices among students, but you are free to use other text editors.

Note: Please, please, *please* do not use word processors such as Microsoft Word to edit programs.

For your reference, we've also written some guides on using popular text editors. After you're done with lab, you can take a look if you're interested:

- Atom (</articles/atom.html>)
- Sublime Text 3 (</articles/sublime.html>)
- Emacs (</articles/emacs.html>)
- Vim (</articles/vim.html>)

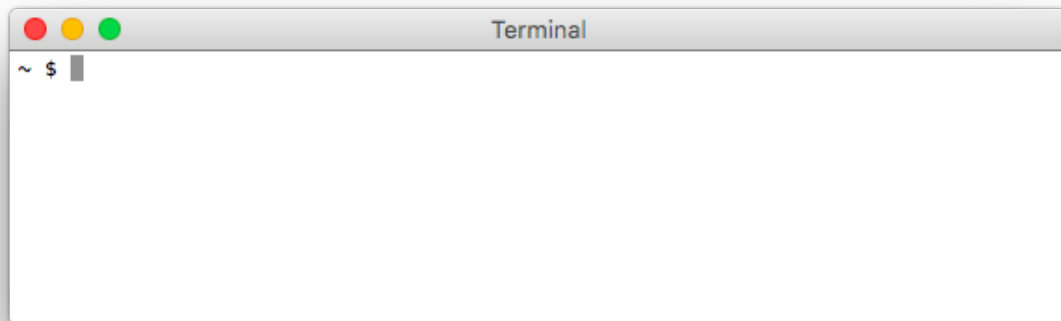
Using the terminal

Let's check if everything was installed properly!

First, open a terminal window. (If you're on Windows, launch *Git-Bash* from the Start menu.)

Important: You should see a `$` sign, waiting for you to type commands.

If you see a `>` sign (such as `C:\Users\0ski>`), you are **not** running Bash; you are in the Windows Command Prompt! Do *not* use that! Close it, and launch Git-Bash instead. (Do *not* launch Git-CMD.)



When you first open your terminal, you will start in the home directory. The **home directory** is represented by the `~` symbol.

Don't worry if your terminal window doesn't look exactly the same; the important part is that the text on the left-hand side of the `$` has a `~` (tilde). That text might also have the name of your computer.

If you see your home directory instead of `~`: Try running `echo ~`. If it displays the same path, that's also fine.

Python Interpreter

We can use the terminal to check if your Python 3 interpreter was installed correctly. Try the following command:

```
python3
```

If the installation worked, you should see some text printed out about the interpreter followed by `>>>` on its own line. This is where you can type in Python code. Try typing some expressions you saw in lecture, or just play around to see what happens! You can type `exit()` or `Ctrl-D` to return to your command line.

Windows troubleshooting:

- If the `python3` command doesn't run *at all*:
Try `python`, `py`, or `py -3` instead.
- If Python freezes (doesn't display anything at all):
You probably didn't select the **"Use Windows' default console window"** option when installing Git-Bash manually. Try `wintty python`, or just uninstall Git-Bash and reinstall it with the correct options.
- If you see an error like `WindowsApps/python: Permission denied`:
Either (a) go to the `WindowsApps` folder whose path is shown and just delete `python.exe` and `python3.exe` in that folder, or alternatively, (b) follow these steps (<https://superuser.com/a/1461471>). Then try again.
- If Python doesn't run *at all*, and you used our automated installer:
Go back and try installing using the manual method.
- If Python doesn't run *at all*, and you installed manually:
Make sure you set up your "PATH" correctly as shown above.
- If you mixed multiple versions of Python (e.g. 32-bit and 64-bit, or 3.6 and 3.8, etc.):
They may conflict. Occasionally, this becomes extremely difficult to fix—even for instructors.
Uninstall them one-by-one (the most recent one *first*), then reinstall only the latest 64-bit version.

Ask for help if you get stuck!

Organizing your files

In this section, you will learn how to manage files using terminal commands.

Make sure your prompt contains a `$` somewhere in it and does not begin with `>>>`. If it begins with `>>>` you are still in a Python shell, and you need to exit. See above for how.

Directories

The first command you'll use is `ls`. Try typing it in the terminal:

```
ls
```

The `ls` command **lists** all the files and folders in the current directory. A **directory** is another name for a folder (such as the `Documents` folder). Since you're in the home directory right now, you should see the contents of your home directory.

If `ls` doesn't work, but `dir` does: **stop!** You've mistakenly opened the Windows Command Prompt!
Exit it, and run *Git-Bash* instead.

Changing directories

To move into another directory, use the `cd` command. Let's try moving into your Desktop directory. First, make sure you're in your home directory (check for the `~` on your command line) and use `ls` to see if the Desktop directory is present. Try typing the following command into your terminal, which should move you into that directory:

```
cd Desktop
```

Although, on some Windows accounts, your actual Desktop folder might actually be inside OneDrive:

```
cd OneDrive/Desktop
```

If you still can't find your Desktop directory, ask a TA or a lab assistant for help.

There are a few ways to return to the home directory:

- `cd ..` (two dots). The `..` means "the parent directory". In this case, the parent directory of `cs61a` is your home directory, so you can use `cd ..` to go up one directory.
- `cd ~` (the tilde). Remember that `~` means home directory, so this command will always change to your home directory.
- `cd` (`cd` on its own). Typing just `cd` is a shortcut for typing `cd ~`.

You do not have to keep your files on your Desktop if you prefer otherwise. Where you keep your files locally will not affect your grade. Do whatever is easiest and most convenient for you!

Making new directories

The next command is called `mkdir`, which **makes** new **directories**. Let's make a directory called `cs61a` on your Desktop to store all of the assignments for this class:

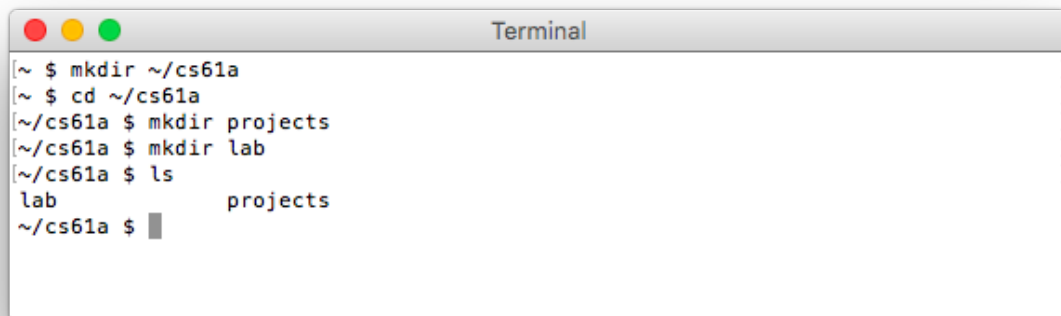
```
mkdir cs61a
```

A folder named `cs61a` will appear on your Desktop. You can verify this by using the `ls` command again or by simply checking your Desktop.

At this point, let's create some more directories. First, make sure you are in the `~/Desktop/cs61a` directory. Then, create folders called `projects` and `lab` inside of your `cs61a` folder:

```
cd ~/Desktop/cs61a
mkdir projects
mkdir lab
```

Now if you list the contents of the directory (using `ls`), you'll see two folders, `projects` and `lab`.

A screenshot of a macOS Terminal window titled "Terminal". The window shows a series of commands and their outputs. The user starts at the home directory (~), then navigates to ~/cs61a. They create two subdirectories, 'projects' and 'lab', and then use the 'ls' command to list the contents of ~/cs61a, which shows 'lab' and 'projects' as directories.

```
[~ $ mkdir ~/cs61a
~ $ cd ~/cs61a
~/cs61a $ mkdir projects
~/cs61a $ mkdir lab
~/cs61a $ ls
lab      projects
~/cs61a $
```

Downloading the assignment

If you haven't already, download the zip archive, `lab00.zip` (`lab00.zip`), which contains all the files that you'll need for this lab. Once you've done that, let's find the downloaded file. On most computers, `lab00.zip` is probably located in a directory called `Downloads` in your home directory. Use the `ls` command to check:

```
ls ~/Downloads
```

If you don't see `lab00.zip`, ask a TA or lab assistant for help.

Extracting starter files

You must expand the zip archive before you can work on the lab files. Different operating systems and different browsers have different ways of unzipping. If you don't know how, you can search online.

Using a terminal, you can unzip the zip file from the command line. First, `cd` into the directory that contains the zip file:

```
cd ~/Downloads
```

Now, run the `unzip` command with the name of the zip file:

```
unzip lab00.zip
```

You might also be able to unzip files without using the terminal by double clicking them in your OS's file explorer.

Once you unzip `lab00.zip`, you'll have a new folder called `lab00` which contains the following files (check it out with `cd` and `ls`):

- `lab00.py` : The template file you'll be adding your code to
- `ok` : A program used to test and submit assignments
- `lab00.ok` : A configuration file for `ok`

Moving files

Move the lab files to the lab folder you created earlier:

```
mv ~/Downloads/lab00 ~/Desktop/cs61a/lab
```

The `mv` command will **move** the `~/Downloads/lab00` folder into the `~/Desktop/cs61a/lab` folder.

Now, go to the `lab00` folder that you just moved. Try using `cd` to navigate your own way! If you get stuck, you can use the following command:

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

Summary

Here is a summary of the commands we just went over for your reference:

- `ls` : **l**ists all files in the current directory
- `cd <path to directory>` : **c**hange into the specified **d**irectory
- `mkdir <directory name>` : **m**ake a new **d**irectory with the given name
- `mv <source path> <destination path>` : **m**ove the file at the given source to the given destination

Finally, you're ready to start editing the lab files! Don't worry if this seems complicated -- it will get much easier over time. Just keep practicing! You can also take a look at our UNIX tutorial (</articles/unix.html>) for a more detailed explanation of terminal commands.

Python Basics

Expressions and statements

Programs are made up of expressions and statements. In very simple terms, an *expression* is a piece of code that evaluates to some value and a *statement* is one or more lines of code that make something happen in a program.

When you type a Python expression into the Python interpreter, its value will be printed out. As you read through the following examples, try out some similar expressions in your own Python shell, which you can start up by typing this in your terminal:

```
python3
```

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

You'll be learning various types of expressions and statements in this course. For now, let's take a look at the ones you'll need to complete today's lab.

Primitive expressions

Primitive expressions only take one step to evaluate. These include numbers and booleans, which just evaluate to themselves.

```
>>> 3
3
>>> 12.5
12.5
>>> True
True
```

Names are also primitive expressions. Names evaluate to the value that they are bound to in the current environment. One way to bind a name to a value is with an assignment statement.

Arithmetic expressions

Numbers may be combined with mathematical operators to form compound expressions. In addition to `+` operator (addition), the `-` operator (subtraction), the `*` operator (multiplication) and the `**` operator (exponentiation), there are three division-like operators to remember:

- Floating point division (`/`): divides the first number by the second, evaluating to a number with a decimal point *even if the numbers divide evenly*
- Floor division (`//`): divides the first number by the second and then rounds down, evaluating to an integer
- Modulo (`%`): evaluates to the positive remainder left over from division

Parentheses may be used to group subexpressions together; the entire expression is evaluated in PEMDAS order.

```
>>> 7 / 4
1.75
>>> (2 + 6) / 4
2.0
>>> 7 // 4          # Floor division (rounding down)
1
>>> 7 % 4           # Modulus (remainder of 7 // 4)
3
```

Assignment statements

An assignment statement consists of a name and an expression. It changes the state of the program by evaluating the expression and *binding* its value to the name in the current frame.

```
>>> a = (100 + 50) // 2
```

Note that the statement itself doesn't evaluate to anything, because it's a statement and not an expression. Now, if we ask for `a`'s value, the interpreter will look it up in the current environment and output its value.

```
>>> a
75
```

Note that the name `a` is bound to the *value* 75, *not* the expression `(100 + 50) // 2`. **Names are bound to values, not expressions.**

Doing the assignment

Unlocking tests

One component of lab assignments is *unlocking* tests. Their purpose is to test your understanding and make sure you have a good enough grasp on the topic to make progress on the assignment.

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.html#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, equivalent to the remainder of 7 // 2
```

```
-----
```

```

>>> x = 20
>>> x + 2

-----

>>> x

-----

>>> y = 5
>>> y += 3                # Equivalent to y = y + 3
>>> y * 2

-----

>>> y /= 4                # Equivalent to y = y // 4
>>> y + x

-----

```

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:

```

def twenty_twenty():
    """Come up with the most creative expression that evaluates to 2020,
    using only numbers and the +, *, and - operators.

    >>> twenty_twenty()
    2020
    """
    return (1 + 10 ** (1 + 1)) * ((1 + 1) ** (1 + 1 + 1) + 1 + 1) * (1 + 1)

```

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: "if we input this Python code, what should the expected output be?"

In `twenty_twenty`,

- The docstring tells you to "come up with the most creative expression that evaluates to 2020," but that you can only use numbers and arithmetic operators + (add), * (multiply), and - (subtract).
- The doctest checks that the function call `twenty_twenty()` should return the number 2020.

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.

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 2020. 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.

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, you should see a successful test:

```
=====
Assignment: Lab 0
Ok, version v1.11.1
=====

~~~~~

Running tests

-----

Test summary
    1 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

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

# Error: expected
#     2020
# 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).

Submitting the assignment

Now that you have completed your first CS 61A assignment, it's time to turn it in. Note that it is **not** sufficient to receive credit for an assignment simply by running the autograder per the last section. You must follow these steps to submit 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.html#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
Backup 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.html>). 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/3.4/using/cmdline.html>).

- Using no command-line options will run the code in the file you provide and return you to the command line.

```
python3
```

- `-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.

```
python3 -i
```

- `-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).

```
python3 -m doctest
```

CS 61A (/)

[Weekly Schedule \(/weekly.html\)](/weekly.html)

[Office Hours \(/office-hours.html\)](/office-hours.html)

[Staff \(/staff.html\)](/staff.html)

Resources (/resources.html)

[Studying Guide \(/articles/studying.html\)](/articles/studying.html)

[Debugging Guide \(/articles/debugging.html\)](/articles/debugging.html)

[Composition Guide \(/articles/composition.html\)](/articles/composition.html)

[Policies \(/articles/about.html\)](/articles/about.html)

[Assignments \(/articles/about.html#assignments\)](/articles/about.html#assignments)

[Exams \(/articles/about.html#exams\)](/articles/about.html#exams)

[Grading \(/articles/about.html#grading\)](/articles/about.html#grading)