

Installing a terminal

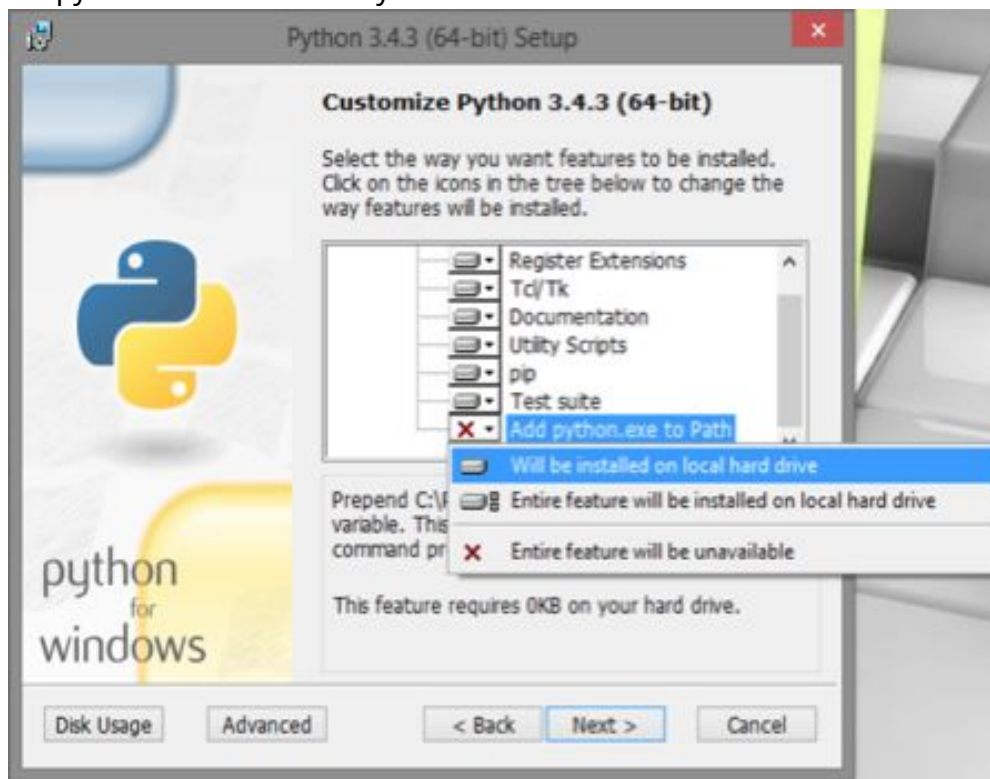
Before we get our hands on our first computer program, we need to set up a few things! First we need to install a terminal. You're probably used to using your computer by clicking on things with your mouse, whether it be opening an application on your desktop or navigating through your file explorer. The terminal itself is a program that allows you to interact with your computer and do all of the same things (run programs, find and move files, etc.), but with only text commands. We will run our code and interact with our program using a terminal.

- If you're on a Mac or are using a form of Linux (such as Ubuntu), you already have a program called Terminal on your computer! To open that up you can search "terminal" in your Spotlight Search (command + space).
- For Windows users, we recommend downloading a terminal called [GitBash](#), or [Cmder](#) (download the full version).

Installing Python 3

The programming language that we will be using for this lab today is Python (version 3.5.1). You can download the files needed to use and run Python [here](#). Make sure you download Python 3.5.1.

- If your computer is a 64-bit machine, you should download the 64-bit installer. (In general, if your computer is new within the past 2 years, it is likely 64-bit. Ask if you're not sure!)
- MacOS users can refer to this [video](#) for additional help on setting up Python. It's a bit old, but might be still helpful. If you're having trouble opening the installer, you can right-click the icon and select "Open".
- For Windows users, if you're installing a more recent version of Python, you should make sure to specify during setup to 'Add python.exe to Path' (picture below is from the installation of Python 3.4.3), which will allow you to execute the python command from your terminal.



- If you did not see the aforementioned option during setup, then you will need to manually configure your PATH environment variable; this [video](#) describes how to do this from 5:00 to 5:54.
- You can also refer to the same [video](#) for additional help on setting up Python (up to 1:09 into the video).

Installing a text editor

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

A text editor is a program that allows you to edit text files, and often comes with tools to help you customize your experience. You will be using a text editor to create, modify, and save files.

There are many editors out there, each with its own set of features. We find that [Sublime Text 2](#) and [Atom](#) are popular choices among students, but you are free to use other text editors. (Atom is the hot new thing right now.)

Note: Please, please, please do not use Microsoft Word to edit programs. Word is designed to edit natural languages like English — trouble will ensue if you try to write Python with Word!

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 (the last two options are text editors that people use in their terminals):

- [Atom](#)
- [Sublime Text 2](#)
- [Emacs](#)
- [Vim](#)

Command Line 101

Let's open your terminal (a.k.a. command line). Your terminal should start inside your home directory by default. Note that "directory" is just another name for "folder" (for example, the Documents folder). They are often used interchangeably.



We will show you how to do four basic things via command line: listing the content of a directory, making a new directory, changing directories, and moving directories/files.

First, we will list the content of the current directory. Let's type `ls` into the terminal:

```
$ ls
dir00 dir01 dir02 dir04 dir05 dir06
```

You will see all the files and folders located in the current directory.

Next, let's make a new directory named `eecs` with the command `mkdir` :

```
$ mkdir eecs
Alex@SAM ~/UC Berkeley/test
$ ls
dir00 dir01 dir02 dir04 dir05 dir06 eecs
```

A folder called `eecs` will be created in your home directory. You can confirm that this works by using the `ls` command. Formally, the format for this command is `mkdir [directory_name]`.

This will create a new folder/directory with the name inside your current directory.

We use the command `cd` to switch into another directory. Type the following command into your terminal:

```
$ cd eecs
```

This command will move you into your `eecs` folder. If you want to go back to the parent directory, you can use the following command:

```
$ cd ..
```

Now, you should be back in the home directory.

The name `..` is used to refer to the parent directory of your current directory).

The name `.` used to refer to your current directory. Formally, the format for this command is:

`cd [path]`,

where `path` is simply the way to get to your destination, so the path `../eecs/day` would tell the terminal to move to the parent directory, then go into the `eecs` folder, then into the `day` folder (if it existed). Similarly, `cd .` would just tell the terminal to move the current folder, (which wouldn't change anything).

Finally, let's try moving files. Navigate yourself back to the parent directory of `eecs` and make a file named `day`:

```
$ mkdir day
```

If you list out the contents of your current directory, you should get a list of files and folders, two of which should be `eecs` and `day`. Let's now move the `day` directory into the `eecs` directory:

```
$ mv day eecs
```

If you change into your `eecs` directory and use `ls`, you should see `day` inside! The format for this command is:

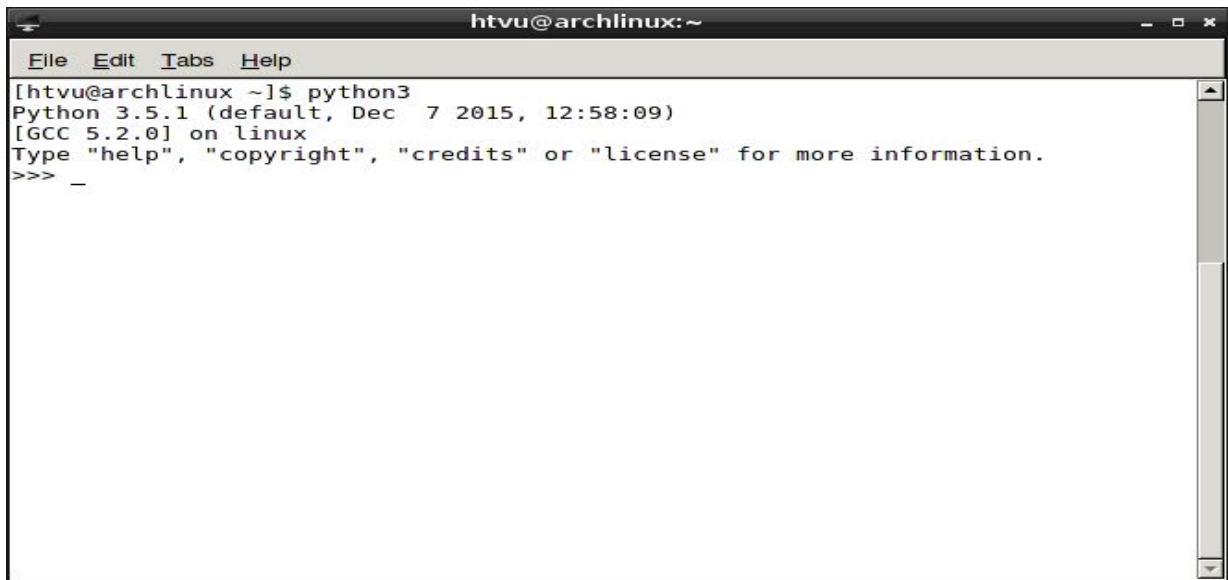
`mv [source_path] [destination_path]`,

where `source_path` is how to get to the thing you want to move, and `destination_path` is where you want to put it. Basically what we told the terminal was move the `day` folder that is in our directory into the `eecs` folder.

Running the Python shell

The Python shell allows you to try out Python commands interactively in real time, line by line. It is usually the place to check out new features or quickly test out your ideas . It's like a special flavor of command line that only runs Python.

To start the Python shell, type `python3` in your terminal. Note, if you are on Windows, you might need to type `python` or `py` instead. Please make sure that the first line printed after your command mentions Python 3.5.1 instead of Python 2.7.x (this is the version of the Python language you are using). We will use Python 3 throughout this lab.



```
htvu@archlinux:~  
File Edit Tabs Help  
[htvu@archlinux ~]$ python3  
Python 3.5.1 (default, Dec 7 2015, 12:58:09)  
[GCC 5.2.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> _
```

Each line beginning with `>>>` is where you can type in Python commands. Let's try a simple calculation, enter the following into the shell: `2 + 2`. You should see the interpreter spit out 4 as the result.

To quit the Python shell and return to the original terminal, type `exit()`.

Running Python scripts

We just learned how to use Python shell interactively in the terminal. Most of the time, however, Python is not run interactively in a shell. Instead, people normally run Python scripts (a.k.a. programs) from their command line. Scripts are just text files that have Python code in them. Instead of typing everything you want to do in the terminal, you can just write everything to a file and have the Python interpreter execute it. This is much easier because you can easily edit and write large programs using a text editor, whereas if you want to run a large program with the Python shell,

you would have to type the whole program each time you wanted to run it (and possibly start over if you make a mistake when typing).

You know [player pianos](#)? Running the Python shell interactively is like playing a song on the piano with your hands. Writing a Python script and running it from the terminal is like making a music sheet roll and then putting it into a player piano and have it play the song for you.

Now let's write a really simple Python script. You'll learn cooler ways to use Python further on in the lab. You can use the text editor you downloaded previously to create and edit Python text files. Open up your text editor and create a new file with the following text:

```
welcome.py
1 print('Hello World!')
2 print(2+2)
3
4 x = 18
5 y = 112
6 z = x * y
7
8 print('Welcome to EECS Day {}!'.format(z))
9
```

Then, save the file in your `eeecs` folder as a Python file with `welcome.py` as the filename. The `.py` file extension (like `.pdf` or `.txt`) tells programs that your file is intended to be a Python script.

In your terminal, navigate to your `eeecs` directory and check that `welcome.py` is in the folder. Then, run the command `python3 welcome.py`. This tells the terminal to run the interpreter `python3` and give it `welcome.py`. You should see the following output in your terminal.

```
$ python3 welcome.py
Hello World!
4
Welcome to EECS Day 2016!
```

Congratulations, you just ran a Python script! Now let's dive into learning how to use the Python language!

What is Python?

From Wikipedia - Python is a high-level, general purpose, interpreted, dynamic programming language that supports object-oriented, imperative, and functional programming, featuring dynamic typing and automatic memory management, emphasizing code readability, unlike this sentence.

What did I just read?

For now, none of that is important. All you need to know is that underneath all the cables and circuits, a computer is a very simple machine. You give it instructions, it follows the instructions, and it gives you the answer. Writing code is just giving the computer a list of instructions.

Imagine you are in a chemistry lab. All around you are different types of chemicals, beakers, burets, bunsen burner and much more. To make a chemical reaction, you need a set of procedures which tell you how you should use the things in the lab to get the results that you want. Similarly, a computer needs code to tell it how to do what you want it to do. Python is just one of the many languages which does this.

Intro to Variables, Assignment, String, Integers

All applications, games, search engines use and store different types of data in [Variables](#). A [Variable](#) stores a piece of data, and gives it a specific name. For example:

The following code creates and assigns the [Variable my_name](#) to "Oski" and the [Variables my_age](#) to 147 and the [Variables my_major](#) to 'EECS'

```
>>> my_name = 'Oski'
>>> age = 147
>>> my_major = 'EECS'
```

Notice how Oski is in quotes while 147 is not. This is because in the Python syntax, [Strings](#) must be surrounded with quote, to differentiate them from other [Integers](#) and [Variables](#).

Once we assign values to [Variables](#), we can call the [Variables](#) and it will give us its value back.

```
>>> my_name
'Oski'
>>> age
147
>>> my_major
'EECS'
```

We can also re-assign the [Variables](#) to a different value and also the values of other [Variables](#)!

```
>>> my_name = 'Berkeley'
>>> my_name
'Berkeley'
>>> my_name = my_major
>>> my_name
'EECS'
>>> my_major
'EECS'
>>> age
147
```

Notice how [my_major](#) is still 'EECS' because we never changed it!

Your turn!

Open up an interactive session in the python shell.

1. Create and assign the variable `name` to a `String` that is your name and the variable `age` to an `Integer` that is your age.
 2. Re-assign `age` to `name`.
 3. Re-assign `name` to a `String` that is your high school's name.
 4. Call `name` to make sure that it is your high school's name and `age` to make sure that it is your name.
-

Booleans

There are other types of variables besides words and numbers. What about a statement, like "Blue Whales are the heaviest mammals?" This statement has only two possibilities - it is `True`, or it is `False`. Statements that are either true or false, with no in-betweens, are called `Boolean` expressions.

Here are a few more examples:

True	False
"Fluorine is the most electronegative element"	$3 + 5 = 7$
$25 \geq 48 / 2$	"Pineapples grow on trees"
"EECS is cool"	"Vikings wore horned helmets"

Think you understand?

Your turn!

Are these expressions `True` or `False`?

"The Campanile is 307 feet tall"

`True`

"Gravity is stronger than the electromagnetic force"

`False`

$12 < 4 + 5$

`False`

"The oldest recorded animal is a giant tortoise"

`True`

Boolean Expressions in Python

How do you ask a computer if an expression is `True` or `False`? Can you just type it in? Try it!

Open your interactive shell and type in

```
>>> 3 + 5 = 7
```

What did you get? An error, right? Why do you think that happens? Think about it from the computer's perspective. How can it tell the difference between `age = 147` - telling the computer that age is 147 - and `age = 147` - asking the computer if age is 147? It can't. The way you get around that is by using two equals signs (`==`) when you are asking for a comparison.

Try this in your interactive shell

```
>>> 3 + 5 == 7
```

Now it's working. What do the rest of these expressions return? Take a guess, then try typing them in. Can you figure out what `<=` and `>=` do?

```
>>> 10 == 3 + 7
True
>>> 5 * 3 < 12
False
>>> 4 * 3 >= 12
True
>>> 5 + 2 <= 6
False
>>> True
True
```

Try coming up with a few examples of your own!

Boolean Operators

There's one more thing to learn about booleans. How do you check if two expressions are both true? How do you check one function or the other? How do you check if an expression is false?

You and your friend are going to Memorial Glade for a picnic. You only want to go if it's sunny **and** the temperature is over 70 degrees. If you were asking your friend, the question could be phrased "Is it warm **and** sunny?" Asking the computer is the same - just use `and` between two expressions.

```
>>> sunny = True
>>> temp = 75
>>> sunny and temp >= 75
```

True

`and` is called a boolean operator. It works just like the mathematical operators (+ - * /) you are used to. Put them between two expressions, and get one back. There are two more operators in Python, called `or` and `not`. `or` is `True` if the first expression **or** the second expression is `True`. `not` doesn't work like and or or. It only takes one expression, and is `True` if the expression is `False`, and `False` if the expression is `True`. Confused? Try it out!

```
>>> 11 == 4 + 5 and 11 > 10
False
>>> 10 > 12 or 5 == 2 + 3
True
>>> not 10 < 11
False
>>> 3 > 4 or 5 == 6 or True
True
>>> not (5 + 2 == 7 and 12 <= 15)
False
```

Notice the parenthesis? They work the same way as in math - you evaluate the expressions inside the parenthesis first, then follow order of operations. By the way, the order of operations is

- `not` first
- `and` second
- `or` last

Remember all that? Try evaluating these. Come up with an answer first, then check it with the interpreter.

```
>>> True or True and False
True
>>> not True and True or False
False
>>> not (((True and False) or (True or False)) and True)
Solution - False
```

There are two extra topics you can consider. In Python, things that are not boolean expressions have a "truthiness" value. When you put something "truthy" in a boolean operator, it acts like `True`. Similarly, something "falsy" acts like a `False`. For example, `1` is a truthy value, and `0` is a falsy value.

```
>>> 1 and True
True
>>> 0 or False
```

```
False
>>> 1 or 0
1
```

Notice that the last one returns `0`, not `False`. That's caused by Python being lazy, or **short-circuiting**. It's an interesting topic - if you're curious, look it up! Be careful though, it's not the same as your phone short-circuiting when you drop it in the pool.

Back to truthiness - can you figure out the truthiness of the following expressions?
How can you test them?

```
2
Truthy
None
Falsy
'Berkeley'
Truthy
"
Falsy
```

Control Flow, Recursion, And Challenge Problem

Go to: <https://gist.github.com/bthananjeyan/68cc9791230ca6bf9430>