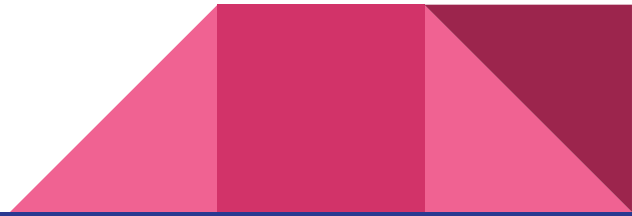
You, Me and Python Env

Getting a handle on the advanced features of Python

Overview

- What is This Lecture?
- Python Virtual Environments?
- What can Python do for me?
- Googling 101

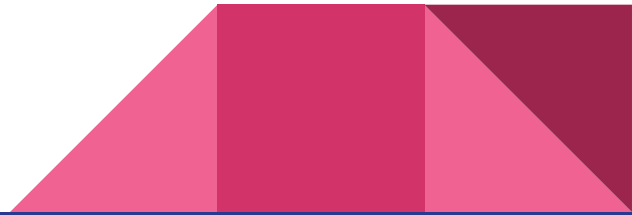


What is this Lecture?

- Python 2 : Electric Boogaloo
  - Python Virtual Environments
  - Some Pythonic Style Conventions
- Google-Fu
  - A few Google commands that are helpful from place to place
  - A few things to keep in mind when trying to find information

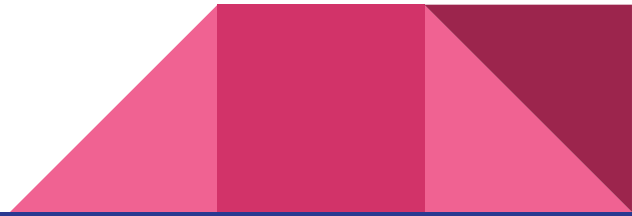
Python Virtual Environments

- Using Python as your default language can be difficult
- Installing outside packages can require certain versions of other packages
- Getting two python installations to use the same software can be hard!
- Enter - Python venv



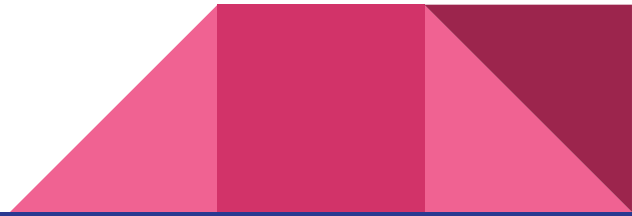
Python venv

- Python’s native tool to install virtual environments
- Allows you to have a “separate” Python installation
  - Each has different packages
  - Each is separate from each other
  - Allows for easy sharing through “requirements.txt” documents



Creating your own virtual environment

- `python -m venv env`
  - Creates a virtual environment without any packages at `./env`
  - Contains a copy of the Python interpreter and standard library
  - Contains no external packages, like `matplotlib` and `pandas`
- `source env/bin/activate`
  - Activates the current Python venv
  - Switches the `python` command to the Python interpreter in the virtual environment
- `deactivate`
  - Deactivates the current Python venv
  - Switches it back to the system installation

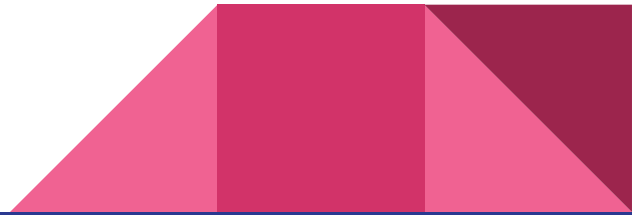


Common Pitfalls with Virtual Environments

- Upgrade pip, setuptools, and wheel before installing more
  - `pip install --upgrade pip setuptools wheel`
- Don’t commit `./env` to your Git repository
  - Commits the Python interpreter and associated binaries to the repo
  - Binaries not universal to all OS
- Don’t use Anaconda’s python to create a virtual environment
  - Sets the `PYTHONPATH` environment variable
  - Messes with standard installation of virtual environments
  - Instead, use `/usr/local/bin/python3 -m venv env`

Sharing Python Virtual Environments

- Instead of sharing the `/env` folder, share `requirements.txt`
  - List of all of our Python packages we installed in our local environment
  - Much, much, much smaller than the full `/env` folder
  - Generate it via `pip freeze > requirements.txt` with your current environment activated
- Installing a python virtual environment using `requirements.txt`
  - `pip install -r requirements.txt` from within a fresh virtual environment



Pythonic Code

What is “Pythonic Code”?

- Clean, readable code that looks like someone who knows Python wrote it.
  - Generally uses Python’s advanced features
  - Tends to be more readable
  - Tends to be more reliable
- Tends to be obvious to users of Python
  - But not so obvious to users of other programming languages
- Leads to code that you can be more proud of
  - Simpler code tends to be easier to read, no matter how bad the logic might be

Why Should You Care?

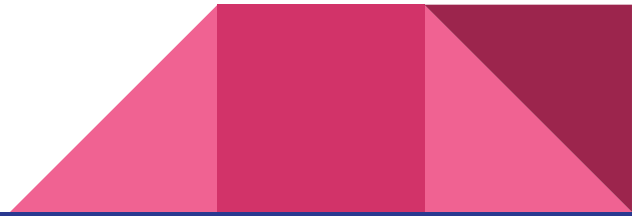
- Many of you learned C++ as your first language
  - Great language for many things
  - Not the be all end all of language design
  - How you code in C++ will impact how you code in other languages
- What feels more natural?

```
my_list = [1,2,3,4,5,6]
for element in my_list:
    print(my_list)
```

```
my_list = [1,2,3,4,5,6]
for i in range(0,6):
    print(my_list[i])
```

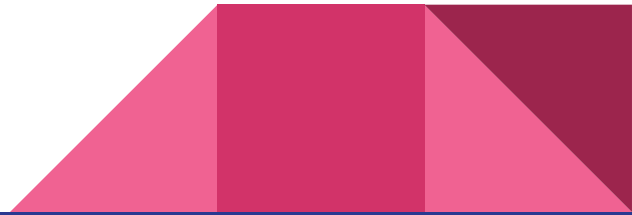
Why Should You Care?

- What language you use a lot will impact how you view problems and how you solve them.
- Lots of people fall into the trap that “all languages are roughly the same”
  - If I spoke Fluent Spanish, I could get by in Italy
  - Does that mean I speak Italian?
  - JS had to implement a “class” keyword to get classes to behave as users expect
- Learning a language in depth requires not just knowing the basics
  - Learn the style and the recommended ways to do things
  - Generally leads to more readable and more efficient code
  - See: R with vectorization.



Why Am I Teaching This?

- There are lots of Python Tutorials out there
  - I've tried courses from Codecademy, Coursera, FreeCodeCamp, DataCamp
  - I've taken courses that each teach Python differently here at U of M
    - 485, 388, 445, 442, etc.
- Most simply teach the basics
  - Very few will talk about the Walrus Operator
  - Even fewer will explain why
- I'll give you some of the intermediate tips, and give some resources for more.



Pythonic Code - List Comprehensions

Python and “Pythonic” Code

- Believe it or not, but Python has a lot to make code easier to write
- Full of features that aren’t taught in most classes, but are useful
  - List comprehensions, but in more depth
  - Tuple Unpacking
  - Advanced Dictionaries
  - Decorators
  - Like the last Python lecture - this is only a taste of what’s available
  - (Refer to either the Python docs or “Fluent Python” for more!)

List Comprehensions

- Being lazy is great!
- Common to loop through a list to change data points, one by one
- List Comprehensions allow us to do this, lazily!
- General Syntax
  - [<operations on item> for item in <list-like object>]
  - Use only if you intend on generating a List
    - Use map if you only care about side-effects, like printing to stdout
    - Use filter if you want to remove values based on some condition

List Comprehensions - Example 1

```
# Let's say we want to get the squares of the first 10 numbers into a list
# We can do it using the following for loop:
input_list = list(range(10)) # Generate a list from 0 to 9
output_list = []
for i in input_list:
    output_list.append(i**2)

# Instead, we can do the following:
output_list = [i**2 for i in list(range(10))]
```

List Comprehensions - Example 2

```
# Let's say we want to generate a list of 5 lists, each of length 6, with values -1
# We can do it using the following nested for loop:
output_list = []
for i in range(5):
    inner_list = []
    for j in range(6):
        inner_list.append(-1)
    output_list.append(inner_list)

# Instead, we can do the following:
output_list = [[-1 for j in range(6)] for i in range(5)]
```

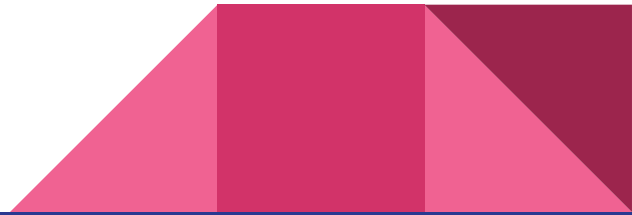
List Comprehensions - Example 3

```
# We can even use if statements in list comprehensions!
# If we want to get only the squares of the odd numbers
# from 1 to 100, we can do it using the following nested for loop:
output_list = []
for i in range(100):
    if i%2 == 1:
        output_list.append(i**2)
# Instead, we can do the following:
output_list = [i**2 for i in range(100) if i%2 == 1]
```

Pythonic Code - Tuple Unpacking

Tuple Unpacking

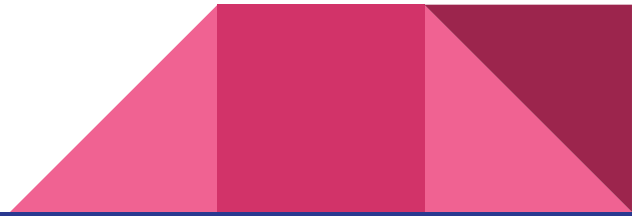
- More of a Python hack, but leads to way cleaner code
- Recall that tuples are a “data structure of ordered fields”
  - a_tuple = (1,2,3)
  - Seemed relatively useless
    - Can’t edit them
    - Can’t extend them
  - While they have their normal uses, seemed relatively esoteric
- However, the truth is a LOT of pythonic code relies on tuples!



Tuple Unpacking

Tuple Unpacking

- Python generates tuples whenever it sees something like the following:
  - a,b = b,a
    - Unique construct to Python.
    - Python's does a swap of the two variables
  - How?
    - Python evaluates the RHS first
      - Creates a tuple (b,a)
    - Python then assigns each value in the tuple to the LHS
      - a gets assigned the past value of b
      - b gets assigned the past value of a



Tuple Unpacking

- Not unique to swaps, either!
  - Does anyone got any ideas what this does?
    - `a, b = 1,1`
    - `a, b = b, a+b`
    - `a = 1,1,2,3,5,8,.....`
- Leads to code that’s easier to read and work on
  - Less temp variables
  - Less noise on what’s happening in between

Tuple Unpacking

- Even useful for your functions!
- Let’s say you want to take any number of arguments

```
def allTheArgs(*argv):
    for arg in argv:
        print("I have you now, ", arg)


allTheArgs("Peter Pan", "Wendy", "Tinker Bell")
```

- Here, * is a term known as the “splat” operator

The “Splat” Operator

- Term coined from Ruby
- Serves to “unpack argument lists and tuples”
- Example usage:

```
a, *b, c = (1,2,3,4)
# a = 1
# b = (2,3)
# c = 4
```

Pythonic Code - Advanced Dictionaries

Advanced Dictionaries

- Deep inside the standard library, there is a package called collections
  - Contains tools that some might seem … useful
  - Tend to make certain tasks easier
    - Counting objects
    - Assigning values to objects
    - Maintaining queues with keys
  - Also lead to faster interview code, if you need to write code for a screener!

Advanced Dictionaries - Counter

- Do you want to simply count the number of objects that you process?
- Introducing collections.Counter
  - Create via c = Counter(iterable)
    - Iterable is any python type you can loop over
      - Lists
      - Dicts
      - Etc.
    - C[key] returns the number of occurrences of key in iterable
  - Can do much more than simply count!

Advanced Dictionaries - Counter

- c.elements()
  - Returns the elements of iterable, returned in the order first encountered
  - Repeats as many times as value occurs
- c.most_common([n])
  - Returns a list of the n most common elements in iterable
- c.update(next_iterable)
  - Processes elements in next_iterable one by one
  - Adds them to the counter
- You can even add and subtract Counters!

Advanced Dictionaries - DefaultDict

- Do you want your dictionary to have default values for missing entries?
- Enter: collections.defaultdict
  - Acts like any normal dictionary in all but missing data cases
  - Creates an entry for the missing key, and assigns it a value you decide
- Example:

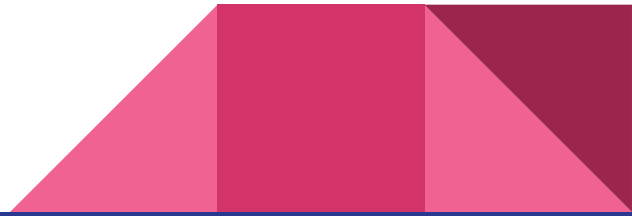
```
d = defaultdict(lambda: -1)
# d["a"] -> -1
```

Advanced Dictionaries - ?????

- The rest of collections is cool, but more application specific
  - RTM!
- See something you want in a Python dict, but it's not there?
  - Create it!
  - Python dicts are just fancy Python classes that implement the following methods
    - Look into "dunder methods", for more information!
    - E.g. __setitem__ , __getitem__
    - If it walks like a dict, talks like a dict, then by golly it's a dict!

Resources for Further Python Reading

- Fluent Python
  - The Definitive Guide to Advanced Python
  - Covers a lot of the class-based stuff that didn't make it into this presentation
- O'Reilly and Manning's Publications on Python
  - Look for what interests you first and foremost
  - O'Reilly gives free access to UMich students through lib.umich.edu
- Hitchhiker's Guide to Python
  - https://docs.python-guide.org/
  - Good reference for what is Pythonic and what is not
  - Also goes in more detail on virtual environments



Google-Fu!

Bonus Round - Google Tips and Tricks!

- Generally useful to learn when you're trying to find information
- Googling is the most important tool you have as a CS major
  - Don't forget that you can get better at it
  - Don't forget that there are good ways to do it and bad ways to do it
- Some tricks are either poorly documented or don't work as well as they used to

Important Keywords

- Searching within a site? Use the site: keyword!
  - Search within SO using site:stackoverflow.com
  - Search within the PyTorch docs using site:pytorch.org/docs/stable
- Searching for a particular filetype? Use the filetype: keyword
  - Look for PDFs with filetype:PDF
  - Especially good when looking for certain information...
- Searching for an exact error message?
  - Remember to put it in quotes “ “
  - Helps you find the SO post you’re looking for

Important Operators

- Use - to screen out words that you do not want
  - If you're googling for a Python error and see a lot of C++, add -"C++"
  - Also helpful when you end up looking for interdisciplinary information
- Use related: to find websites similar to certain websites
  - Good for finding similar resources, or when you have an idea of what you're looking for
  - Related:google.com lists a bunch of other search engines
- Use cache: to get Google's version of websites
  - Helpful when the website is down or impossible to access
  - Alternative to archive.org that sometimes includes stuff archive.org doesn't have

The Most Important Google Tool

- You
  - Understanding the problem leads to solving the problem
  - Spend 90% of your time on thinking about the problem before spending time solving it
  - Better Google Searches come from critically thinking about what you're looking for
- Trust, but Verify Information
  - Many, many cases where a poor google search results in bad information
  - Make simple cases where something should work before using it in production
  - Forcing yourself to understand WHY the error happens helps ensure you remember the solution itself
    - "Moonwalking with Einstein" <- Terrific Book on Memory and What Sticks

That’s All!
Good Luck on Finals!