Recitation Week 3

PRANUT JAIN

Administration

Recording issue with Zoom for last week's recitation

Webpage: https://people.cs.pitt.edu/~pranut/TA/CS1520-Spring21/

Plan for Today

Python Setup

Python basics

Quiz!

Setting up Python(Windows)

Download and install from https://www.python.org/downloads/

The latest version is 3.9.1

- Add to Windows path during installation, can also be done later manually
- (Recommended) Install PIP during installation, useful for downloading python packages.
- Can leave other options to default.

Python Virtual Environments

The main purpose of Python virtual environments is to create an isolated environment for Python projects.

This means that each project can have its own dependencies, regardless of what dependencies every other project has.

So, why do all of these little details matter?

By default, every project on your system will use these same directories to store and retrieve site packages (3rd party libraries).

Doesn’t affect packages part of standard python library, but **site** packages are affected.

Setting up Python VE

Creation of virtual environments is done by executing the command venv:

python3 -m venv /path/to/new/virtual/environment

Running this command creates the target directory (creating any parent directories that don't exist already) and places a pyvenv.cfg file in it with a home key pointing to the Python installation from which the command was run.

It also creates a bin (or Scripts on Windows) subdirectory containing a copy of the python binary (or binaries, in the case of Windows). It also creates an (initially empty) lib/pythonX.Y/site-packages subdirectory (on Windows, this is Lib\site-packages).

c:\>python -m venv c:\path\to\myenv (Windows, assuming PATH variable has been setup)

Python Reference

https://docs.python.org/3/tutorial/introduction.html

Quiz and python slides based on the above tutorial.

Go through it for in-depth understanding of various Python constructs.

Python Argument Passing

Using argv variable.

When -m module is used, sys.argv[0] is set to the full name of the located module.

sys.argv[0] otherwise has the script name, the rest of the list has arguments [1] and so forth.

Lists

Can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

```
>>> squares = [1, 4, 9, 16, 25]

>>> squares

[1, 4, 9, 16, 25]
```

Like strings (and all other built-in sequence type), lists can be indexed and sliced.

Indexing Lists

```
>>> squares[0]  # indexing returns the item
1
>>> squares[-1]
25
>>> squares[-3:]  # slicing returns a new list
[9, 16, 25]
```

Appending to Lists

You can also add new items at the end of the list, by using the append() method (we will see more about methods later):

```
>>> cubes = [1, 8, 27, 65, 125]

>>> cubes.append(216)  # add the cube of 6

>>> cubes.append(7 ** 3)  # and the cube of 7

>>> cubes

[1, 8, 27, 64, 125, 216, 343]
```

Python Loops

Python’s for statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

```
>>> # Measure some strings:
... words = ['cat', 'window', 'defenestrate']
>>> for w in words:
...     print(w, len(w))
...
cat 3
window 6
defenestrate 12
```