External Modules

15-110 – Monday 11/25

Learning Goals

Learn how to **install and use** external modules

Identify **common useful modules** for the Python language

Python Modules

The Python programming language comes with a large set of build-in functions that cover a range of different purposes. However, it would take too long to load all of these functions every time we want to run a program.

Python organizes all of its different functions into **modules**. When you run Python, it loads only a small set of functions from the build-in module. To use any other functions, you must **import** them.

Built-in Modules

We've already used a few core modules for homework assignments - **math, tkinter,** and **random.**

We've introduced other useful modules in lecture- **copy** (for breaking aliases), **csv** (for parsing CSV files), **json** (for parsing json files), **glob** (for finding files with common names), and **datetime** (storing information about dates).

For a full list of python libraries, see here. Let's briefly introduce one other useful module - **os**.

os module

The **os** module lets you directly interact with your computer's **operating system**. The operating system is the part of the computer that manages the low-level operations, like deciding which program gets to run on the processor, and deciding where data is stored in memory.

You can use the os module to modify files on your computer. The following functions are especially useful:

- os.listdir(path) # returns a list of files in the directory
- os.path.exists(path) # returns True if the given path exists
- os.rename(a, b) # changes file a's name to b
- os.remove(path) # deletes the file. WARNING: deleting via program is permanent!

External Modules

There are many other libraries that have been built by developers outside of the core Python team, to add additional functionality to the language. These modules don't come as part of the Python language, but can be added in. We call these **external modules**.

In order to use an external module, you must first **install** it on your machine. This means you'll need to download the files from the internet to your computer, then integrate them with the main python library, so that the language knows where the module is located.

Finding Useful Modules

One of the main strengths of Python as a language is that there are thousands of external modules available, which means that you can start many projects based on work others have done instead of starting from scratch.

You can find a list of popular modules here and a more complete list of pip-installable modules here. We'll go over a few of the most popular among CMU students in the next few slides.

pip

It is usually possible to install modules manually, but this process can be a major pain. Luckily, python also gives us a streamlined approach for installing modules- the **pip module**! This feature can locate modules that are indexed in the Python Package Index (a list of commonly-used modules), download them, and attempt to install them.

Traditionally, we don't run pip from our normal editor- instead, you'll need to run it from the **terminal**. This is a command interface that lets you make changes directly to your computer. On Mac and Linux machines, you can find the terminal by searching your applications for the built-in app Terminal. On Windows, search for the built-in application Powershell.

Running pip

To run pip in the terminal, we use the command:

```
pip install [module-name]
```

This will identify the module and start the download and installation process. It may run into a **dependency error** if the module needs a second module to already be installed- in general, installing that module and then running pip again will fix the problem.

Note: you will not be able to run pip on CMU cluster machines, as these have restricted permissions. You may need to log into your main account on personal machines to run it.

Using an Installed Module

Once you've successfully installed a module, you should be able to put

```
import [module-name]
```

at the top of a python file, and it will load the module the same way it would load a built-in library!

Note: this may fail if you have multiple versions of Python installed on your machine. Make sure to use the pip associated with the version of Python you're using in your editor. You can check your editor's version in Pyzo with Shell > Edit Shell Configurations (check the value in exe), then call pip using

```
python-[version number] -m pip install [module-name]
```

Learning how a Module Works

Once a new module is installed, you're still left with one major question: how do you use it?

This varies by module, but the best answer is to **read the documentation**. Most external modules have official documentation or APIs that describe which functions exist and how to use the module.

It can also be helpful to search online for other projects that have used the same module, to find examples of how to set it up. Many people have written helpful tutorials online for this exact purpose.

Two standard resources for finding help are **StackOverflow**, a site where people can ask questions about code and get answers from other developers, and **GitHub**, a site where people post open-source projects for others to use and contribute to.

Reminder: always cite others' work!

You'll sometimes find a useful bit of code in a StackOverflow post or a GitHub project that you'll want to use in your own project.

Whenever you copy code from online, make sure to **cite it** the same way you would cite a paragraph of text in an essay. You can do this by putting a comment above the copied code that includes a link to the URL you got the code from.

This serves two purposes. First- it gives credit to the individual who originally wrote the code. Second- if you run into a problem with the code later on, you'll be able to look back to the original source to find a solution.

Note: policies around copying code change when you're working on a commercial product. Read the fine print if you're planning to sell your code!

Useful Modules

SciPy Collection

SciPy is a group of modules that support advanced mathematical and scientific operations. It can handle large calculations that might take the default Python operations too long to compute. We've already shown you a bit of this in the data analysis lectures!

The group includes NumPy (which focuses on math), SciPy (science), pandas (data analysis), and Matplotlib (plotting of charts and graphs). These can be used separately or as a group. Each need to be installed separately, but can be installed directly with `pip install [name]`

Website: https://www.scipy.org/

SciPy Collection Example

We've already showcased how to use pandas and Matplotlib. Here's a brief demo of running a t-test with Numpy and Scipy:

```
# Run a T-test on two random sets of data
import numpy, scipy
from scipy import stats
vals1 = numpy.random.random(1000)
vals2 = numpy.random.random(1000)
result = stats.ttest_ind(vals1, vals2)
print(result.pvalue)
```

scikit-learn

scikit-learn is a module that supports a large set of machine learning algorithms in Python. If you want to dabble in machine learning or artificial intelligence, this is a good place to start. For any machine learning algorithm, though, you'll need to provide a starting dataset to get it to work.

Website: https://scikit-learn.org/stable/

Install:

```
pip install scikit-learn
```

scikit-learn Example

```
# Learn a decision tree from a random set of two-number data points that predict a third number
import sklearn
from sklearn import tree
import matplotlib.pyplot as plt

trainingX = [ numpy.random.random(2) for i in range(1000) ]
trainingY = numpy.random.random(1000)

regr = tree.DecisionTreeRegressor(max_depth=2)
regr.fit(trainingX, trainingY)

plt.figure()
tree.plot_tree(regr)
plt.show()
```

nltk

nltk, the Natural Language Toolkit, assists with natural language processing for machine learning purposes. This is useful whenever you're working with a corpus of written texts.

Website: https://www.nltk.org/

Install:

```
pip install nltk
```

nltk Example

```
# Identify the nouns in a document
import nltk
document = "insert example text here"
result = []
words = nltk.word_tokenize(document)
tags = nltk.pos_tag(words)
for tup in tags:
    (word, type) = tup
    if (word.lower() not in result) and (type == 'NN' or type == 'NNS'):
        result.append(word.lower())
result.sort()
print(result)
```

Beautiful Soup

Beautiful Soup is a module that supports webscraping and HTML parsing. This is useful if you want to gather data from online for use in an application.

Website: https://www.crummy.com/software/BeautifulSoup/

Install:

```
pip install beautifulsoup4
```

Beautiful Soup Example

```
from bs4 import BeautifulSoup

import requests
page = requests.get("https://www.cs.cmu.edu/~110/schedule.html")
soup = BeautifulSoup(page.content, 'html.parser')
for link in soup.find_all('a'):
    url = link["href"]
    if "slides/" in url and ".pdf" in url:
        print(link["href"])
```

PyAudio

PyAudio makes it possible to analyze, create, and play audio files. This module requires some complex pre-existing software, including the language C++; if you get an error message while installing, read it carefully to see how to make the installation work.

Website: https://people.csail.mit.edu/hubert/pyaudio/

Install:

```
pip install pyaudio
```

Note that there are many other audio modules available as well; you can find a list here.

PyAudio Example

```
import pyaudio, math
# Make the sound data
bitrate, freq = 64000, 130.815 # C3 frequency
dataFrames = [""] * 5
for octave in range(5):
    mult = 2*math.pi*(freq*(octave+1))
    for frame in range(bitrate):
        dataFrames[octave] += chr(int(math.sin(mult * frame / bitrate) * 127 + 128))
# Play the sounds!
p = pyaudio.PyAudio()
stream = p.open(format=32, channels = 1, rate = bitrate, output = True)
for frame in dataFrames:    stream.write(frame)
# Close the stream
stream.stop_stream()
stream.close()
p.terminate()
```

Django

Django is a module that lets you build interactive websites using Python. This involves setting up a **frontend** (the part of a website that the user sees while browsing) and a **backend** (the part of a website that processes requests and does the actual work).

Website: https://www.djangoproject.com/

Install:

```
pip install django
```

Fun Modules

PIL: Python Imaging Library

PIL is a lightweight and easy-to-install module that lets tkinter interact with images other than .gif and .ppm files. It also includes functions that support basic image manipulation.

Website: http://www.pythonware.com/products/pil/

Since the main PIL installation is not maintained, most programmers use an offshoot called Pillow instead.

Website: https://pypi.org/project/Pillow/2.2.1/

Install:

```
pip install pillow
```

Pygame

Pygame is, like tkinter, a library that lets you make graphical applications. However, Pygame is specifically designed to create games. It has better support for sprites and collision detection than tkinter. Pygame uses an event loop similar to ours, which makes it fairly easy to learn.

Website: https://www.pygame.org/news

Install:

```
pip install pygame
```

Panda3D

Panda3D is a module that supports 3D rendering and animation. Like pyaudio, it can be very complicated to install and use, but it is still much easier than trying to create 3D animation in a 2D system.

Website: https://www.panda3d.org/manual/

Install:

```
pip install Panda3D
```

Learning Goals

Learn how to **install and use** external modules

Identify **common useful modules** for the Python language