External Modules

15-110 – Bonus Slides

Module Index

Math: NumPy, SciPy

Data Analysis: Matplotlib, pandas

Machine Learning: scikit-learn

Computer Vision: OpenCV

Natural Language Processing: nltk

Websites: Django, Flask

Webscraping: Beautiful Soup

Images: Pillow

Audio: Pydub

Game Design: Pygame

3D Graphics: VPython

Data Analysis External 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.

The group includes NumPy (which focuses on core math), SciPy (math and science functions), 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/

NumPy

NumPy's main purpose is to support mathematical operations in Python.

That may not seem necessary at first, since Python already has support for many math operations in the built-in libraries, but NumPy has the advantage of being very efficient; that makes it a great library to use on large datasets.

NumPy Arrays

NumPy mostly works as you would expect for a Python library. Its main difference is that it organizes numbers in lists differently from regular Python.

NumPy creates special array objects of varying dimensions (like one- and two-dimensional lists); these arrays can represent vectors or matrices in mathematical calculation.

```
import numpy as np

np.array([10, 20, 30, 40])
# [10 20 30 40]
np.array([ [1, 2, 3], [4, 5, 6],
           [7, 8, 9] ])
# [[1 2 3]
#  [4 5 6]
#  [7 8 9]]
```

NumPy Arrays – Operations

NumPy supports a lot more built-in operations on arrays that Python does on lists. For example, you can directly add a number to an array; that will add the number to each of the numbers inside the array.

You can also directly subtract one array from another; that will take the difference of the numbers at matching indexes.

```
import numpy as np

a = np.array([10, 20, 30, 40])

b = np.array([5, 6, 7, 8])

a + 2 # [ 12 22 32 42 ]
a - b # [ 5 14 23 32 ]
```

NumPy Arrays - Indexing

In addition to all of this, NumPy also supports more advanced indexing into multi-dimensional arrays.

For example, if you make a 2D array, you can index into it with row comma col instead of needing to use two indexes. This can be quite handy!

```
import numpy as np

c = np.array([ [1, 2, 3],
    [4, 5, 6], [7, 8, 9]])

c[1, 2] # 6
```

NumPy Functions

NumPy does have its own set of mathematical functions- for example, it can generate random numbers.

```
x = numpy.random.randint(1, 10)
# random number in [1, 10]
```

However, it's mostly used as a support library for other libraries that want to use more efficient mathematical operations when doing statistics, or science, or engineering tasks.

SciPy

- Next, let's look at the SciPy library. SciPy provides functions for scientific computation, mostly relying on NumPy for lower-level calculations.

- The SciPy functions tend to be a bit higher-level so that you can run whole processes automatically instead of scripting them yourself.

- SciPy also splits its functions into different sub-libraries. For example, there's a sub-library called linalg for linear algebra, one called signal for signal processing, and one called stats for statistics.

SciPy Functions

SciPy isn't too hard to use when you have a specific purpose in mind. Just find the sub-library that corresponds to what you want to do, set up your data properly, and run the function.

For example, if you want to find the inverse of a matrix (where your matrix times its inverse equals the identity matrix), there's a function for that!

Just import the `linalg` sub-library, set up your matrix as a NumPy two-dimensional array, then run `linalg.inv`.

```
import numpy as np
from scipy import linalg

a = np.array([ [ 1, 2 ], [3, 4] ])

linalg.inv(a))
# [[-2.   1. ]
#  [ 1.5 -0.5]]
```

Matplotlib

The **matplotlib** library can be used to generate interesting visualizations in Python. This is great for data analysis!

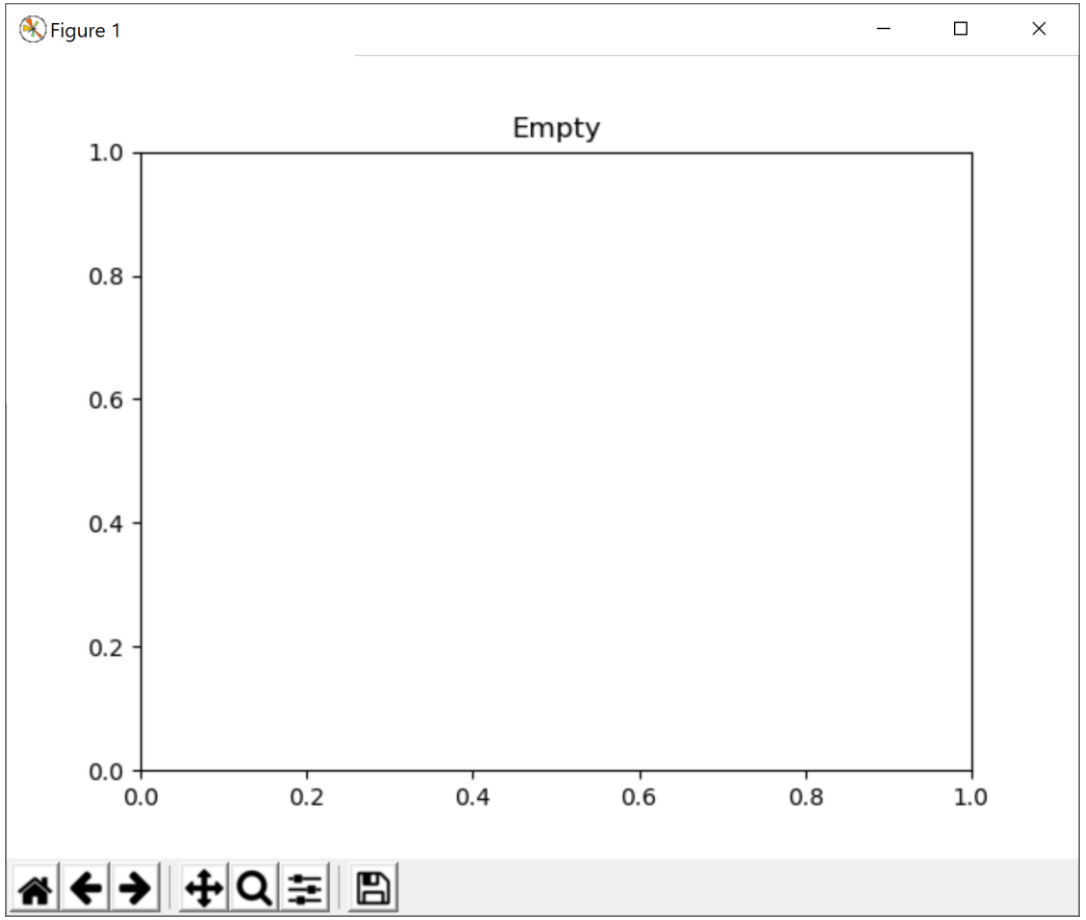
The way that specific types of charts and graphs are set up can vary a lot, but there are some core components to every chart that are consistent.

Draw Visualizations on the Plot

Matplotlib visualizations can be broken down into several components. We'll mainly care about one: the **plot** (called `plt`). This is like Tkinter's canvas, except that we'll draw visualizations on it instead of shapes.

We can construct an (almost) empty plot with the following code. Note that matplotlib comes with built-in buttons that let you zoom, move data around, and save images.

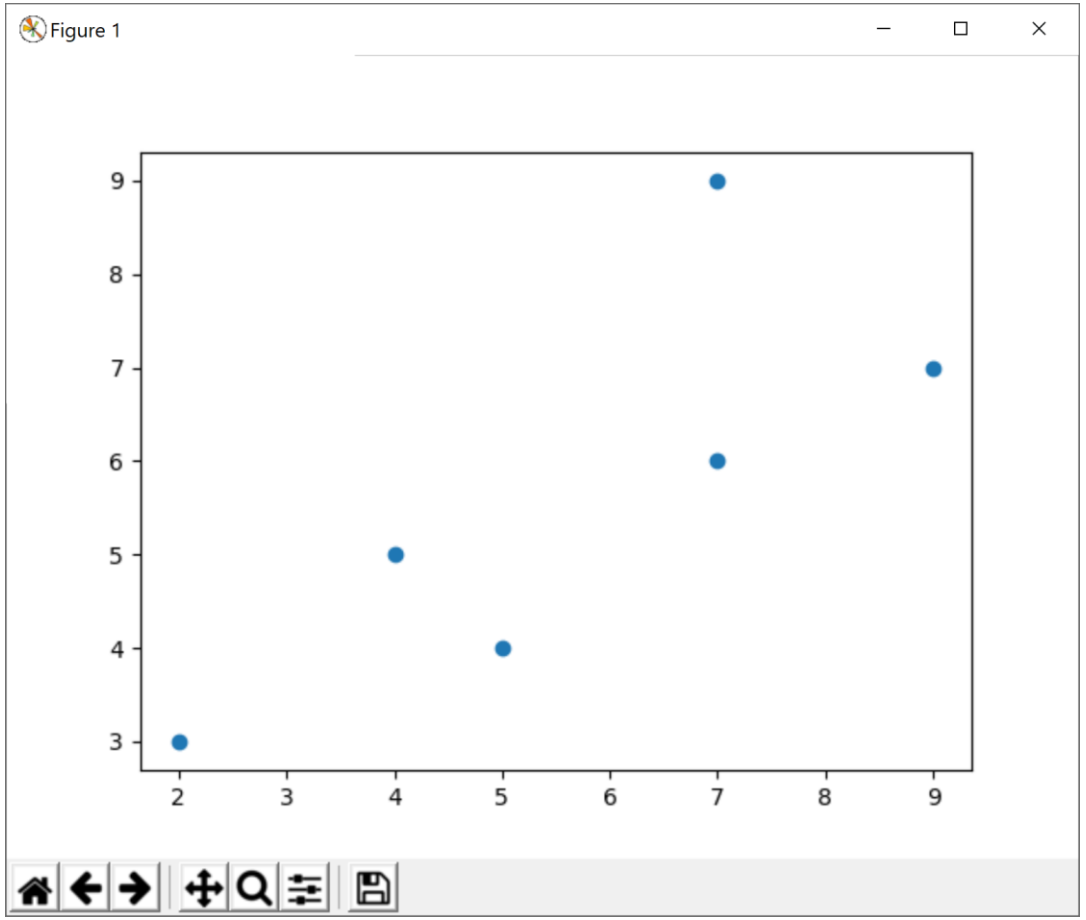
```
import matplotlib.pyplot as plt

plt.title("Empty")
plt.show()
```



Add Visualizations with Methods

There are lots of built-in methods that let you construct different types of visualizations. For example, to make a scatterplot use `plt.scatter(xValues, yValues)`.

```
x = [2, 4, 5, 7, 7, 9]
y = [3, 5, 4, 6, 9, 7]
plt.scatter(x, y)
plt.show()
```

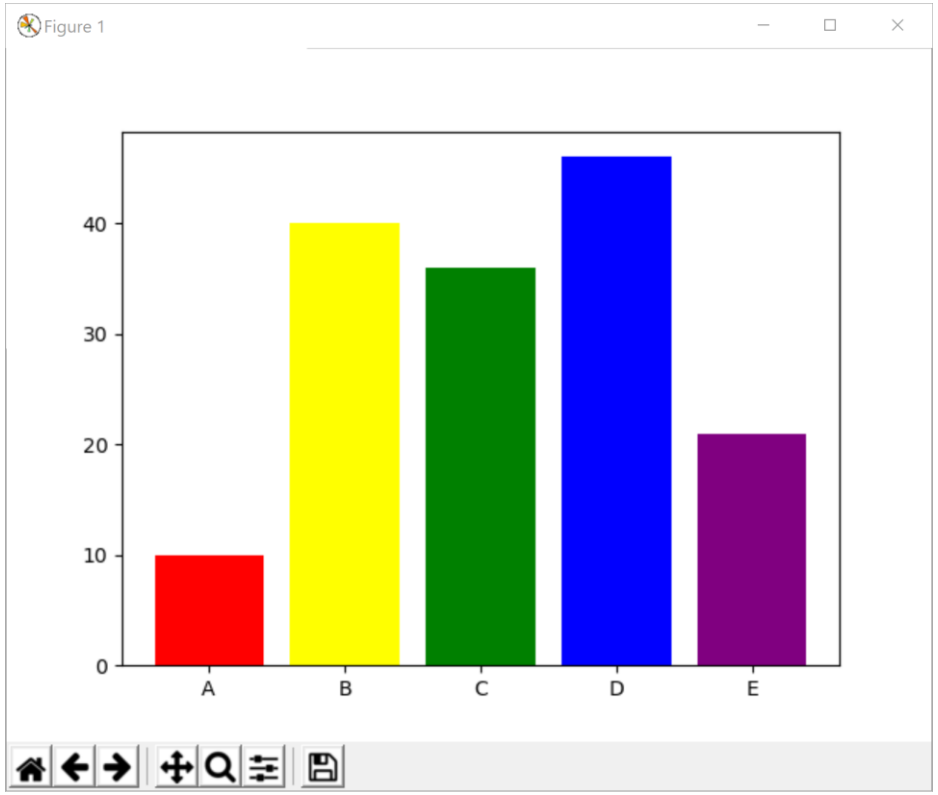


Visualization Methods have Keyword Args

You can **customize** how a visualization looks by adding **keyword arguments**. We used these in Tkinter to optionally change a shape's color or outline; in Matplotlib we can use them to add labels, error bars, and more.

For example, we might want to create a bar chart (with `plt.bar`) with a unique color for each bar. Use the keyword argument `color` to set the colors.

```
labels = [ "A", "B", "C", "D", "E" ]
yValues = [ 10, 40, 36, 46, 21 ]
colors = [ "red", "yellow", "green",
           "blue", "purple" ]
plt.bar(labels, yValues, color=colors)
plt.show()
```



Matplotlib Approaches

If you browse the Matplotlib website, you'll see that charts can be drawn with one of two different approaches - object oriented or procedural.

The object-oriented approach lets you break down the window into objects, then control each object independently. Charts are drawn by calling methods on appropriate objects. To learn more about objects, read here: https://docs.python.org/3/tutorial/classes.html

The procedural approach instead has you call all functions from one central library, `matplotlib.pyplot`, which is usually aliased to `plt`. Charts are drawn by calling functions to set up all the elements you want.

Either approach is generally fine- just pick one and stick with it.

Fig and Ax

Here's a quick thing to know if you're using the object-oriented approach. Every graph is drawn in a figure, which has some number of axes. A figure is like a window that pops up on your screen; an axis is a part of that window dedicated to one specific visualization.

To interact with the figure and axis directly, call `plt.subplots` to access the two objects. This function returns two objects, so you should set up two variables to capture the results. This can be done easily with `fig, ax = plt.subplots()`.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
```

Pandas

Pandas is specifically built to support data analysis for data in spreadsheets, or tables. It's great for Excel-style coding.

Pandas mostly works on a kind of data structure called a DataFrame, which is basically a spreadsheet table.

Pandas DataFrames

DataFrames act a bit like 2D lists, except that it's as easy to access data by column as it is to access data by row.

You can set up a DataFrame directly from a 2D list fairly easily, as is shown here. You can also load a DataFrame directly from a CSV file.

Note that, like in NumPy arrays, the DataFrame is displayed a little differently- there are no commas between values, and the row and column indexes are included directly.

Some properties are familiar, though; for example, the length of a DataFrame is just the number of rows.

```
import pandas as pd

df = pd.DataFrame([ [1, 2, 3],
                    [4, 5, 6] ])

df2 = pd.read_csv("data.csv")

print(df)
#    0  1  2
# 0  1  2  3
# 1  4  5  6

print(len(df)) # 2
```

Pandas DataFrame Columns

Where DataFrames get really interesting is that you don't need to refer to columns by index.

You can give them names instead, just like you often would in a spreadsheet with a header!

We can do this by adding a keyword argument, columns, with a list of column names. Column names are also loaded automatically when a table is loaded from a CSV.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])
print(df)
#   Course Number                                Course Name  # Units
# 0        15-110                     Principles of Computing       10
# 1        15-112  Fundamentals of Programming and Computer Science       12
# 2        15-251       Great Ideas in Theoretical Computer Science       12
```

Pandas DataFrame Indexing

Once you have a DataFrame set up, you can index into it by column with a normal index operation, just by providing the column name.

For example, if we index by "Course Number" in the DataFrame we've created here, we'll get the values in that column of the table. Note that when the values are displayed, they're paired with their row indexes, and the column name and type are shown at the bottom. Handy!

Indexing into a specific row is harder – use `.iloc` with the row's position as an index.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])


print(df["Course Number"])
# 0    15-110
# 1    15-112
# 2    15-251
# Name: Course Number, dtype: object

print(df.iloc[0]["Course Number"]) # Principles of Computing
```

Pandas DataFrame Looping

When you loop over a DataFrame with a for-iterable loop, you loop over the columns, not the rows! This is sort of like how a dictionary maps over keys.

If you want to loop over the rows instead, use the method `df.iterrows`, which produces two values per iteration – the index of the row and the row itself. You can index into a column of a row the same way you can index into a column for the whole dataframe.

It's generally better to do work directly with columns when possible, though.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])


for col in df:
    print(col)
# Course Number
# Course Name
# # Units

for index, row in df.iterrows():
    print(index, row["Course Name"])
# 0 Principles of Computing
# 1 Fundamentals of Programming and CS
# 2 Great Ideas in Theoretical CS
```

Pandas DataFrame Subsets

If you just want to work with a subset of the data in a DataFrame, there are two easy ways to do that.

If you want a particular range of rows, slicing works on DataFrames the same way it does on lists!

Or if you want to select a subset of rows based on a specific property they share in a given column, use a **Boolean operation** to index into the DataFrame instead of a normal index. This will evaluate to a DataFrame containing only the rows where that operation evaluated to `True`.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])


sub1 = df[:2]
print(sub1) # only rows 0 and 1
#   Course Number                          Course Name  # Units
# 0        15-110              Principles of Computing       10
# 1        15-112  Fundamentals of Programming and CS       12

sub2 = df[df["# Units"] == 12]
print(sub2) # only rows where # Units was 12
#   Course Number                          Course Name  # Units
# 1        15-112  Fundamentals of Programming and CS       12
# 2        15-251       Great Ideas in Theoretical CS       12
```

Pandas Functions

Outside of DataFrames, the pandas library works about the way you'd expect. You can call methods on DataFrames to analyze the data in them or modify the table as needed.

```
import pandas as pd

df = pd.DataFrame([ ["15-110", "Principles of Computing", 10],
                    ["15-112", "Fundamentals of Programming and CS", 12],
                    ["15-251", "Great Ideas in Theoretical CS", 12] ],
                    columns=["Course Number", "Course Name", "# Units"])
```

For example, if we want to get the median number of units of all the courses in the dataset, we just need to index into the # Units column, then call the `median` method on that set of data values.

```
print(df["# Units"].median())
# 12.0
```

Machine Learning External Modules

Machine Learning Overview

Machine learning is the process of algorithmically finding patterns in a dataset, so that a machine can answer questions about new data or group similar pieces of data together.

There are many, many different algorithms that have been designed to support machine learning. Most machine learning libraries implement those algorithms for you; all you need to do is decide which algorithm is the best fit for your data

For general machine learning, we'll recommend **scikit-learn**. For specialized algorithms, we'll discuss **OpenCV** and **nltk**.

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. Note that you'll still need to provide a starting dataset to get any algorithm to work.

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

Install:

```
pip install scikit-learn
```

Understanding Algorithms

Running algorithms with scikit-learn isn't too hard; you just call methods on your data to set up a model and then use the model to make predictions or check results as needed.

The harder part of machine learning is understanding how the algorithms work, and knowing which algorithm to use for any given task.

There's no shortcut for this; you just have to do a lot of learning to get familiar with lots of different possible approaches.

We'll cover machine learning in a few weeks, and we'll talk more about how to choose the proper algorithm there.

scikit-learn Demo

Let's just look at one example to see what the general process looks like.

I've got a grade dataset - five quiz grades and a final exam grade for a set of 145 students - and I want to cluster the data points, to see which groups naturally emerge.

I'll use a clustering algorithm for this, and I'll specifically choose to use K-means clustering.

Loading Data from a File

Let's use the built-in csv library to load a spreadsheet into a 2D list (we'll go over how to do this next week).

```
import csv

f = open("grades.csv", "r")
reader = csv.reader(f)
data = list(reader)
f.close()
```

Running the Algorithm

Next, we need to create a model based on the data.

We'll run the KMeans algorithm and tell it to create three clusters. Then we'll fit that model to the dataset. The resulting model is an object with certain properties.

```
from sklearn.cluster import KMeans
```

```
model = KMeans(n_clusters=3).fit(data)
```

scikit-learn Model Properties

For example, you can check what the average scores of the three clusters are by looking at the `cluster_centers_` property. The first five numbers are quiz scores, and the last is a final exam score.

```
model.cluster_centers_
# [[89.4271 94.4223 91.5873 95.2281 91.7184 87.7427]
#  [61.7857 81.7857 48.9285 80.6428 66.4285 63.5   ]
#  [87.1    89.7    66.2857 90.6285 90.7142 79.    ]]
```

scikit-learn Model Properties

How many students are in each cluster? You can check that by looking at the labels_ property. This shows which cluster label was assigned to each data point. By turning the labels list into a list and running the count method, you can check how many students are in each group.

```
model.labels_
# [0 2 0 0 0 2 0 2 0 2 0 0 2 2 0 0 0 0 2 2 0 0 0 0 0 0 0 2 2 0 0 2 2 2 1 2 0
#  0 0 0 2 2 0 0 2 0 0 1 2 0 0 0 2 0 0 0 0 0 1 0 2 0 0 0 0 0 2 0 0 0 0 1 0 2
#  0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 2 0 0 0 0 0 2 0 0 2 2 0 0 0 0 0 0 0 0 2 0 0
#  0 0 0 0 2 0 0 0 0 2 0 0 0 0 2 2 1 0 2 2 0 0 2 0 0 0 0 0 0 0 2 0 0 0]

L = list(model.labels_)
L.count(0) # 103
L.count(1) # 7
L.count(2) # 35
```

scikit-learn Model Functions

Finally, if you want to use this model to put a new data point into one of the clusters, use the predict method.

Predict takes a list of data points, so put the single data point in another list. The result is the cluster that the student is assigned to.

```
student = [60, 70, 75, 80, 85, 87]
model.predict([student]) # [2]
```

OpenCV

The OpenCV library is a good choice if you want to do machine learning with images. CV in this case stands for computer vision.

You can install it under the name opencv-python, then import it with the name cv2. Usually this gets aliased to cv.

```
pip install opencv-python
```

```
import cv2 as cv
```

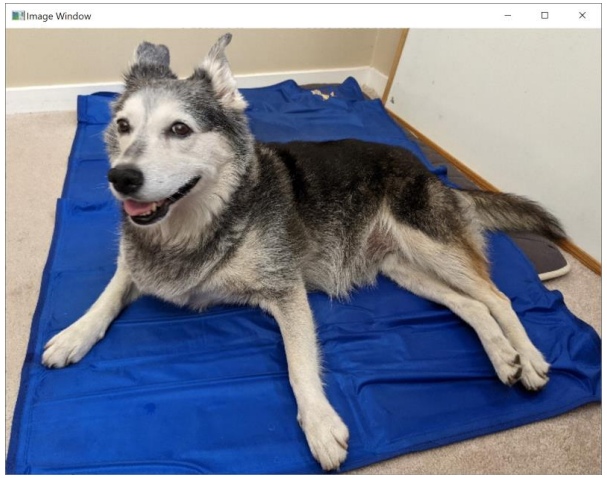
OpenCV Example

OpenCV lets you load images and recognize features in them, like lines, or corners, or digits.

For example, let's say I want to detect edges in an image. First, I can load an image with `imread`.

I can check that image with `imshow` too if I want to! But I have to set up a line of code afterwards so that the program knows to keep the window open and close it when you press x.

```
import cv2 as cv

img = cv.imread("dog.jpg")

cv.imshow("Image Window", img)
k = cv.waitKey(0)
```



OpenCV Example

To detect lines in the image, I need to call a method on the image object. I'll use a Laplacian algorithm and set the depth threshold fairly low.

```
import cv2 as cv
img = cv.imread("dog.jpg")
lines = cv.Laplacian(img, cv.CV_8U)
```

Now we can see that the algorithm automatically detected edges around the dog, and the mat she was laying on. Pretty cool!

```
cv.imshow("Image Window", lines)
k = cv.waitKey(0)
```

Being able to detect these kinds of features makes it easier to run images through machine learning algorithms.

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 Functions

One handy thing you can do with nltk is to tokenize text. This takes a sentence and breaks it up into words.

Note that this doesn't just split up the string by spaces- it intelligently breaks up words based on punctuation as well.

```
import nltk

text = '"My heart is in the work!" Andrew said.'

nltk.word_tokenize(text)
# ['``', 'My', 'heart',
   'is', 'in', 'the',
   'work', '!', "''",
   'Andrew', 'said', '.']
```

nltk Functions

You can also do sentiment analysis with nltk, where you build a model to detect whether a piece of text is generally positive, negative, or neutral based on the words in contains.

You can train your own model, but you can also use a pre-built model by importing the `SentimentIntensityAnalyzer`, which is in the sub-library sentiment.

Then you can just call the method `polarity_scores` on the text you want to score to see what the model thinks!

```
import nltk.sentiment


analyzer = nltk.sentiment.SentimentIntensityAnalyzer()
analyzer.polarity_scores('"My heart is in the work!" Andrew said.')
# {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
analyzer.polarity_scores("Today is such a beautiful day!")
# {'neg': 0.0, 'neu': 0.488, 'pos': 0.512, 'compound': 0.636}
analyzer.polarity_scores("I'm in a bad mood. Go away.")
# {'neg': 0.412, 'neu': 0.588, 'pos': 0.0, 'compound': -0.5423}
```

Web Development External Modules

Regular Web Design

If you want to build a simple website- something that will just display some text and images, perhaps - you should stick to the core languages of website design, HTML for content structure and CSS for styling.

And if you just want to program in a little interactivity, the language Javascript is easy to integrate with HTML and CSS, and is commonly paired with those languages.

But if you want to build a more complex website that keeps track of user state and saves data, Python can help you out!

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
```

Django Principles

The core principles behind Django are that it is object-oriented and it uses databases extensively.

Django uses **objects** to represent the data tracked by a website. For example, a User object might have properties like username, and password, and items in a shopping cart.

A **database** is a data structure that's based on a table. Databases are designed to be able to hold a lot of data and to make it easy to look up data based on specific properties.

Programming with Django

Programming websites in Django is pretty complicated. It's only really needed for complex websites, and you'll need to learn a bit about object-oriented programming before starting.

Once you're ready, you can work through a tutorial of how to set up a Django website here:

https://docs.djangoproject.com/en/3.2/intro/tutorial01/

Flask

Flask is also a module that lets you build interactive websites using Python. But Flask starts with a simple, lightweight website, instead of requiring you to set up a database and objects first.

Website: https://flask.palletsprojects.com/

Install:

```
pip install flask
```

Programming in Flask

You can set up a simple website in Flask pretty easily, just by using a function and some advanced Python syntax.

```
import flask

app = flask.Flask("Example")

@app.route("/")
def tutorial():
    return "<p>My Website</p>"
```

But if you want to make more advanced websites, you'll still need to learn some complex Python syntax first.

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
```

Parse HTML as Tags

HTML organizes content on a page using **tags**, like this:

```
<tag attribute="value">
     <subtag> Some content for the subtag </subtag>
</tag>
```

To parse a website, you need to look for a certain type of tag in the file.

Load HTML with urllib

First, how can you get the HTML of a website?

There's a handy built-in library for that, urllib. It doesn't work in all cases (like when authentication is required), but it works for basic websites.

```
from bs4 import BeautifulSoup
import urllib.request

page = urllib.request.urlopen("https://docs.python.org/3/")
text = page.read()

doc = BeautifulSoup(text, 'html.parser')
```

BeautifulSoup Navigation

You can access the content of a BeautifulSoup page with tag names.

For example, at the top level of the whole document, request the title of the page by accessing the property title, with a period between the variable and the property. Or access the whole body of the page with the property body.

```
doc = BeautifulSoup(text, 'html.parser')
doc.title # <title>3.9.6 Documentation</title>
doc.body # <body><div aria-label="related navigation" ...
```

More BeautifulSoup Navigation

Sometimes tags will have other tags inside them. When this happens, you can continue accessing specific tags at each level.

```
doc.body.p # <p>Welcome! This is the documentation for Python
...
```

For example, to get the first paragraph tag in the body, use `doc.body.p`.

If you want to get the text within that tag, just use the string property.

```
doc.body.p.string # '\n  Welcome! This is the documentation
for ...
```

BeautifulSoup Functions

To get **all** the tags of a certain type in the document, instead of just the first tag of that type, use the method `find_all` on the document. This produces a list of all the tags of that type.

```
a_tags = doc.find_all('a')
a_tags
# [ <a accesskey="I" href="genindex.html" title="General Index">index</a>,
#   <a href="py-modindex.html" title="Python Module Index">modules</a>,
#   <a href="https://www.python.org/">Python</a>,
#   <a href="#">3.9.6 Documentation</a>,
#   ... ]
```

BeautifulSoup Attributes

Note that some of those a tags had properties. You can access the property of a tag with a dictionary index.

```
a_tags[0]["href"]
# "genindex.html"
```

So if I wanted to get all the links on a webpage, I could use this:

```
for tag in a_tags:
    print(tag["href"])
# genindex.html
# py-modindex.html
# https://www.python.org/
# ...
```

Creative External Modules

Pillow: Python Imaging Library

Pillow is a lightweight and easy-to-install module that lets you manipulate images beyond .gif files. It lets you modify images, or use different types of images in Tkinter.

Website: https://pillow.readthedocs.io/en/stable/index.html

Install:

```
pip install pillow
```

Import:

```
import PIL
```

Pillow Images

Pillow makes it very easy to open image files, using the `Image.open` function. You can even display those images with `image.show`, or save them with `image.save`.

```
from PIL import Image
img = Image.open("stella.jpg")
img.show()
img.save("new-stella.jpg")
```

Pillow Image Functions

You can provide Pillow images to the Tkinter `create_image` function, but you can also manipulate them directly!

There are functions that let you crop and resize pictures within the program, and much more! However, some of these functions require that you use lists to hold the dimensions of the picture.

```
newImg = img.crop([200, 200, 3500, 2500])
newImg.save("new-stella.jpg")

newImg2 = img.rotate(180)
newImg2.save("new-stella-2.jpg")

newImg3 = img.resize([1000, 1000])
newImg3.save("new-stella-3.jpg")
```

Pydub

Pydub makes it possible to play and edit audio files! However, it needs a few additional libraries to work really robustly. First, you'll need to install the library simpleaudio to play the edited sounds.

Website: https://github.com/jiaaro/pydub

Install:

```
pip install simpleaudio
pip install pydub
```

Pydub AudioSegments

Pydub lets you load a sound file into an AudioSegment object. It's then really easy to play the sounds and edit them!

Unfortunately, by default the library only supports .wav files. It's possible to get the library to work on more popular filetypes (like .mp3 files), but requires a lot of complicated installations.

First, let's just look at a simple example with a .wav file. Note that once the file starts playing, it will continue until the song ends; you'll need to interrupt the program by pressing the lightning bolt button if you want to stop it early.

```
from pydub import AudioSegment
from pydub import playback

music = AudioSegment.from_wav('song.wav')
playback.play(music)
```

Pydub AudioSegment Editing

To edit music with pydub, you use **slicing**, like how you edit strings and lists.

The AudioSegment represents the song in millisecond segments. So to get the first second of a song, you'd use:

```
song[:1000] # first second
```

Or to get only the first half of a song, you'd use:

```
song[:len(song)//2] # first half
```

Pydub AudioSegment Editing

You can also change the volume of a song by adding or subtracting decibels from it.

This works like NumPy arrays – you can add or subtract a single number from the segment, and it will propagate across all the values.

```
song - 20 # make quieter
```

Pydub AudioSegment Functions

There are a bunch of cool functions already implemented for you. For example, you can implement fading, or speed up a song or remove silence.

Note that these functions may take a little while to run- be patient! You can always save the result in a new file, then play that file directly.

```
music = music.fade_in(10*1000)
music = music.speedup(2)
music = music.strip_silence()

music.export("new_song.wav", format="wav")
```

Pydub and MP3 Files

If you want to edit more popular music file formats (like mp3 files), you've got two options.

One: try to install `ffmpeg`, a non-python library that supports a wide range of audio formats. Unfortunately, you can't do this with pip. Here's instructions from Pydub on how to install: https://github.com/jiaaro/pydub#getting-ffmpeg-set-up

Two: convert your MP3 file into a WAV file using a different audio application. One option is VLC, which is available for free. You can convert files by going to Media > Convert/Save, but you'll need to set up a new profile format for WAV. Here's instructions for how to do that: https://promincproductions.com/blog/export-wav-audio-file-from-any-video-clip-with-vlc/

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`.

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

Install:

```
pip install pygame
```

Pygame Essentials

The core difference between coding a game versus other kinds of coding is that the game needs to be interactive, which means that it needs to keep running continuously while waiting for input from the user.

Pygame supports this through a **game loop**. This is just a while loop that loops until you tell it to stop.

Inside that loop, the game constantly checks for input from the user, responds to any inputs it has received, and generally keeps the game moving.

Pygame Window

Let's say we want to open up a simple window. We can do that with the `set_mode` method, but that by itself isn't enough.

We also want to be able to close that window by pressing the x button. So we need to constantly check whether the user has asked to quit inside the game loop.

We can do this by checking all the events that were received by the game system. When a `QUIT` event happens, exit the loop, then call the built-in function `exit` to exit the window as well.

```
import pygame

pygame.init()

playing = True

width, height = 500, 500
screen = pygame.display.set_mode([width, height])

while playing:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            playing = False

exit()
```

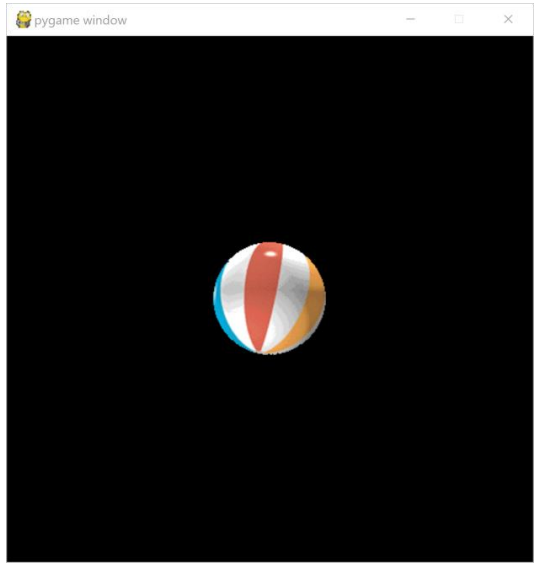
Pygame Demo

This isn't a game yet, though- it's just a window.

Let's make a simple clicker game. The user can click on an image on the screen; whenever they do, their score goes up by one.

This demo will go over some of the core components of Pygame, but there's a lot more it doesn't cover! You can do quite a bit with this library.

Pygame Images



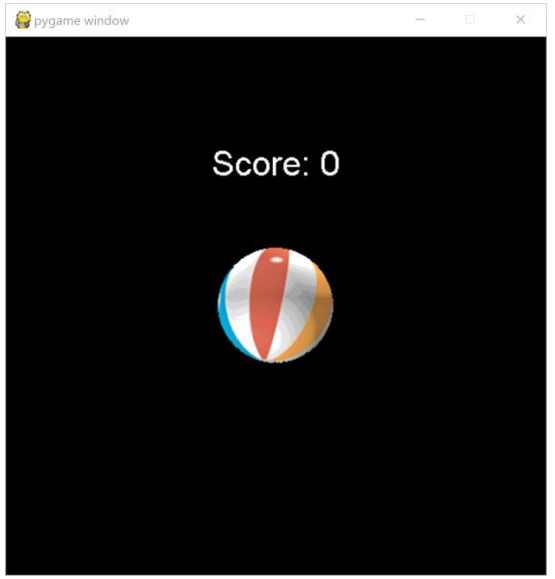
The tricky thing here is that Python needs to refresh the window every time the game loop runs, just in case something changes. So you should actually draw the image inside the game loop.

However, you only want the image to show up once. Before you draw anything, fill the background of the screen with a solid color using `screen.fill`, to erase anything drawn before.

First, load the image using the `pygame.image.load` method. Then set up a rectangle that corresponds to the image in the position where you want to show up. Use `get_width` and `get_height` on the image to make sure it is centered. Finally, use `screen.blit` to actually draw the image in the game loop.

```
import pygame

pygame.init()

playing = True
black = pygame.Color(0, 0, 0)
width, height = 500, 500
screen = pygame.display.set_mode([width, height])

icon = pygame.image.load("ball.gif")
icon_rect = pygame.Rect(250 - icon.get_width()/2,
                        250 - icon.get_height()/2,
                        250 + icon.get_width()/2,
                        250 + icon.get_height()/2)

while playing:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            playing = False

    screen.fill(black)
    screen.blit(icon, icon_rect)
    pygame.display.flip()

exit()
```

Pygame Fonts



You should also set up a score in text. The score itself can just be a variable, and the text can be displayed above the image.

Set up a font first; then render text based on that font. Then the text can just be displayed with `blit`.

This time, let's specifically provide a coordinate pair with the location where we want the text to show up.

```
import pygame

pygame.init()

...

score = 0

white = pygame.Color(255, 255, 255)
font = pygame.font.SysFont("Arial", 32)
score_text = font.render("Score: " + str(score),
                         False, white)

while playing:
    ...

    screen.fill(black)
    screen.blit(icon, icon_rect)
    screen.blit(score_text, [250 -
                score_text.get_width() / 2, 100])
    pygame.display.flip()

exit()
```

Pygame Collisions

Now we need to detect whether the image has been clicked on. The easiest way to do this is with collision detection.

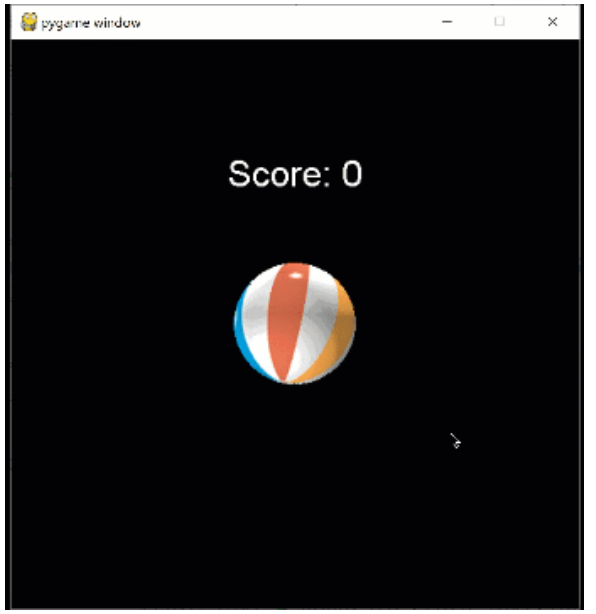
Luckily, this is something that Pygame does really well! If you can capture where the user clicked on the screen, you can easily detect whether that pixel location collided with the image's icon.

To capture the clicked location, check for a new event type - a `MouseButtonUp` event. This will happen when the user has clicked on the screen and releases the button. When this happens, use the `pos` property of the event to get the mouse's current position.

Then call the method `collidepoint` on the image's rectangle and the point to see if they collide. If they do, update the score.

```
import pygame

pygame.init()

...

while playing:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            playing = False
        elif event.type == pygame.MOUSEBUTTONUP:
            if icon_rect.collidepoint(event.pos):
                score += 1
    ...
```

Pygame Collisions



This is a good start, but it isn't enough by itself. We've updated the score, but we haven't updated the score text, so the change will never be registered on the screen.

You need to update the `score_text` variable to show the new score. For now, let's just copy and paste the line we used before. In the future, though, this would be better placed in a **helper function**.

And with that, the game works!

```
import pygame

pygame.init()

...

while playing:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            playing = False
        elif event.type == pygame.MOUSEBUTTONUP:
            if icon_rect.collidepoint(event.pos):
                score += 1
                score_text = font.render("Score: " +
                                         str(score), False,
                                         white)
    ...
```

Game Programming

This approach works, and it's fine if you're not planning to develop the game any further, but if you do plan to extend what the game can do, you should restructure the code a bit.

Object-oriented approaches are really useful for game design. Putting all your major game components into classes helps to organize your code and makes it much easier to manage the core game loop.

This kind of approach also makes it easier to add new features, as it's more clear where the code should go.

Vpython

Vpython is a nice library for creating and interacting with 3D graphics. It's mainly aimed towards scientific simulations, though; for more game-like 3D graphics, you'll probably want to use a game engine (like Unity, or Unreal).

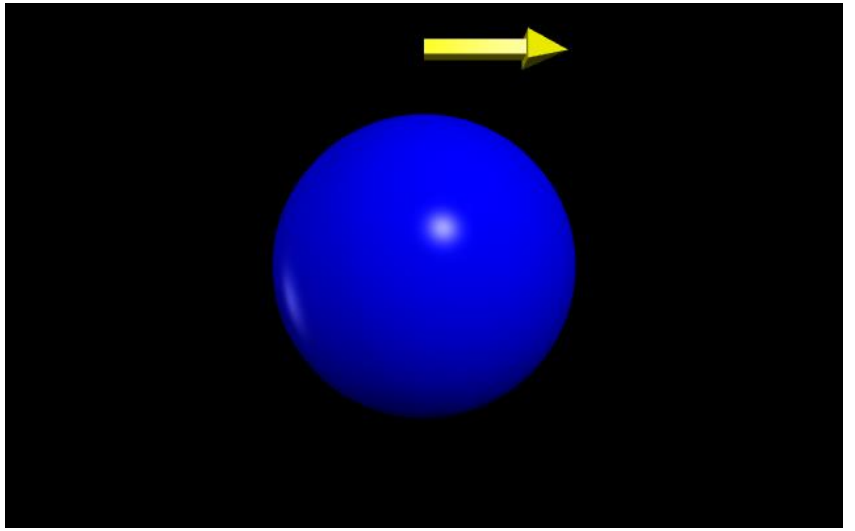
Website: https://vpython.org/

```
pip install vpython
```

Vpython 3D Objects

Programming in Vpython is mostly similar to programming in Tkinter. The main difference is that you construct 3D objects like spheres and points instead of 2D shapes.

```
from vpython import *
ball = sphere(pos=vector(0, 0, 0),
              radius=100, color=color.blue)
pointer = arrow(pos=vector(0, 150, 0),
                axis=vector(100, 0, 0),
                color=color.yellow)
```



When you run the script, it opens a browser window to render the graphics.

Vpython 3D Object Manipulation

You can then program in your own physics to interact with the 3D objects or make them move. Usually this is done by setting up an infinite while loop.

The rate function tells the while loop how long to wait between iterations. This makes it possible to actually see the animation move.

For example, by changing the vector position of the ball, we can make it move back and forth. We can even have it change directions by changing the delta movement.

Changing the axis of the arrow makes it point in the same direction as the ball is moving.

```
move = vector(10, 0, 0)
while True:
    rate(100)
    ball.pos = ball.pos + move
    if ball.pos.x >= 300 or \
       ball.pos.x <= -300:
        move = vector(-move.x, 0, 0)
        pointer.axis = -pointer.axis
```