RECORDING IN-PROGRESS

Day 8: Modules

Wednesday, 7.21.21

RECORDING IN-PROGRESS

Agenda

- Daily check-in
  - Announcements
  - Review solution to Lab 7
- Recap from yesterday
- Look at a little C++ vs. Python code
- New topic: Python Modules & Libraries

Daily check-in

Check in

- Questions?
- Announcements
  - Our tech trek today (National Renewable Energy Lab, or “NREL”) will be from **12:30 - 1:30pm EST**, in Zoom as usual, **guest speaker Kristi Potter from NREL**
  - NREL is a government-funded DoE (department of energy) research lab in the US. NREL focuses on researching and developing renewable energy practices and energy efficient techniques (for our environment! yay!)
      - Crazy fast supercomputer
      - The “CAVE”
    - Fun fact: Ashley interned at NREL during undergrad as a SULI (I highly recommend the SULI program if you’re in the US!)
- Lab 7: Search Algorithms
  - Aaron will go over the solution, please ask him questions

RECAP

Python collections

Python collections (aka **containers** aka **collection data types**) give us a way to store values, or data, in specific ways. Depending on the collection type we use, the data will be stored, accessed, and updated differently.

Examples of Python collections:

- Lists
- Sets
- Tuples
- Dictionaries

Lists

Given the list:

```
    players = [“Mario”, “Luigi”, “Yoshi”, [“Daisy”, “Peach”]]
```

What are some things we know about this list?

- **Is the list ordered?**
    - Yes: our list always starts at the index number 0, and each element is ordered by its position in the list, e.g., 0, 1, 2, 3, ..., n-1
- **Can we access the values in the list?**
    - Yes: we can use the list’s index to access elements directly, e.g., players[0]
- **Can we update the values in the list?**
    - Yes: we can update any value in our list by directly updating the element, e.g., players[0] = “Wario”
- **Can we have duplicate values in the list?**
    - Yes: players = [“Mario”, “Mario”, ...]

Collections

These same questions we just answered about lists:

- **Is the container ordered?**
- **Can we access the values in the container?**
- **Can we update the values in the container?**
- **Can we have duplicate values in the container?**

Is how we differentiate between *all* of the data collections in Python.

We can answer those same questions not just for lists, but also for **sets**, **tuples**, and **dictionaries**. They are each unique in their own way!

Sets

Sets are generally the same across many modern programming languages.

The purpose of a set is to store a collection of values, where no value is repeated twice. In other words, sets do not allow duplicate values.

- **Is the container ordered?** No
- **Can we access the values in the container?** No, can’t use an index
- **Can we update the values in the container?** Sets are considered mutable, but because sets can’t be accessed by their index, we can’t update them
- **Can we have duplicate values in the container?** No

Tuples

Tuples are similar to Lists, except we can’t update or change the values in our Tuples -- unlike Lists. They allow duplicates, and they are ordered.

https://www.w3schools.com/python/python_tuples.asp

- **Is the container ordered?** Yes
- **Can we access the values in the container?** Yes, you can index in using [i]
- **Can we update the values in the container?** No, tuples are immutable
- **Can we have duplicate values in the container?** Yes

Dictionaries

Dictionaries let us store data as a key-value pair, by assigning a unique “key” its own “value”:

```
dictionaryName[key] = value

dictionaryName = {key1: value1, key2: value2, key3: value3}
```

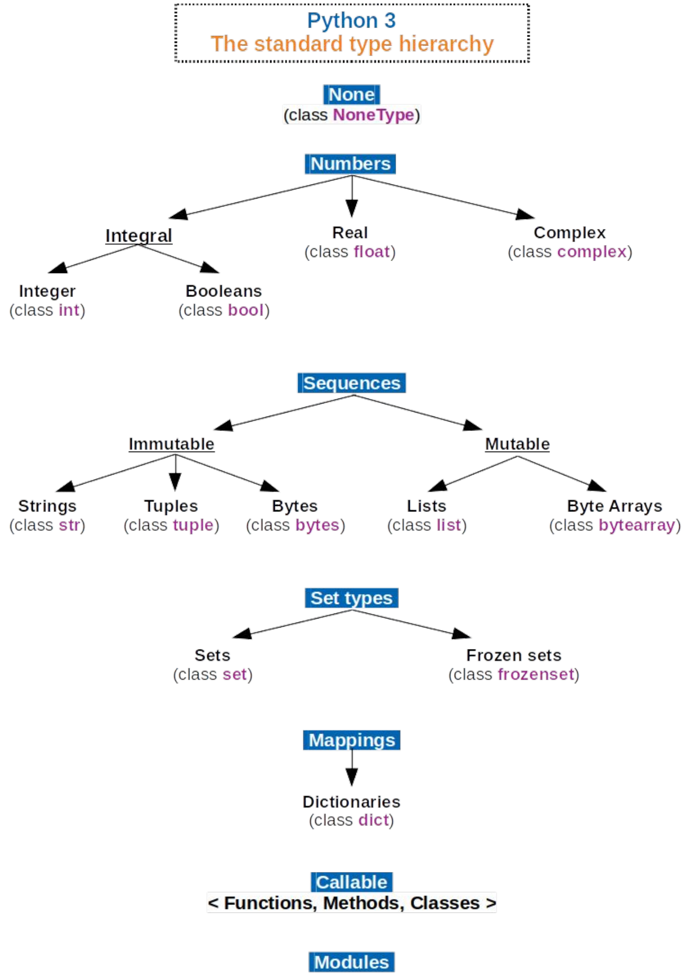
- **Is the container ordered?** No
- **Can we access the values in the container?** Not with an index, but we can get a value from its key
- **Can we update the values in the container?** Not with an index, but we can replace a key’s value
- **Can we have duplicate values in the container?** No duplicate keys, duplicate values ok

Remember this from day 1???

Does this figure look less scary today? :)

Which of these have we learned?

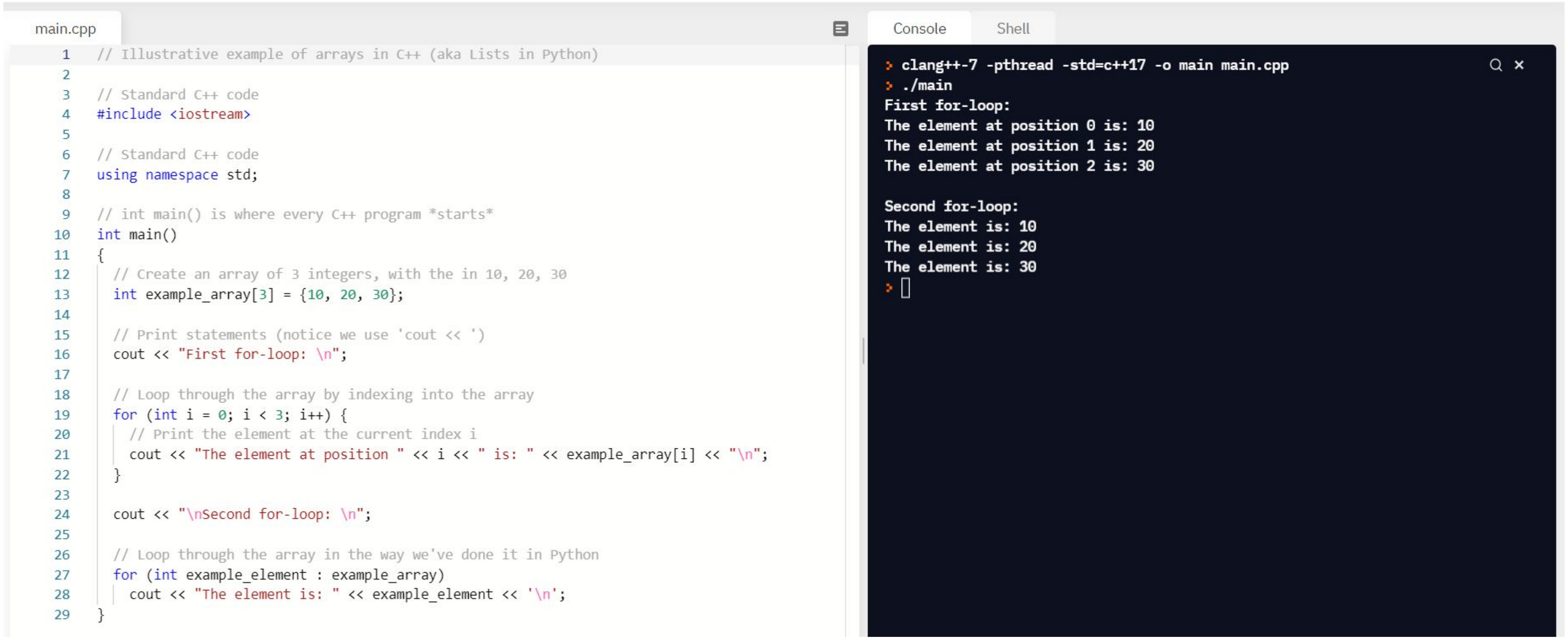
Which haven't we?



Practicing our concepts (C++ vs. Python)

During your first computer science course at college, you will probably use C, C++, or Java. Let’s look at a little C++ code, and see how it compares to Python.

Arrays in C++ = similar to Lists in Python, with a caveat*

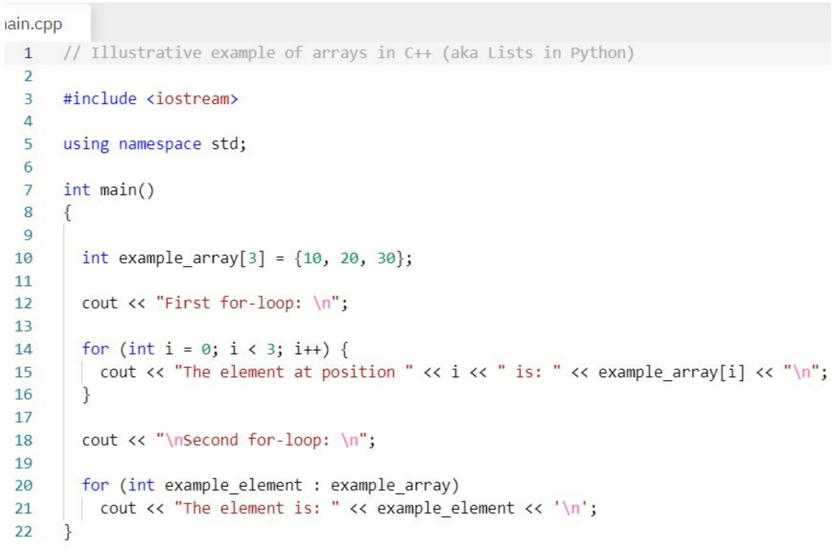

main.cpp

```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

// Standard C++ code
#include <iostream>

// Standard C++ code
using namespace std;

// int main() is where every C++ program *starts*
int main()
{
  // Create an array of 3 integers, with the in 10, 20, 30
  int example_array[3] = {10, 20, 30};

  // Print statements (notice we use 'cout << ')
  cout << "First for-loop: \n";

  // Loop through the array by indexing into the array
  for (int i = 0; i < 3; i++) {
    // Print the element at the current index i
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  // Loop through the array in the way we've done it in Python
  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
>
```

* When using *Arrays* in C++, you *must* specify and maintain the size (i.e., number of elements) of your Array in your code
(int example[3] means your array can contain up to 3 integers)

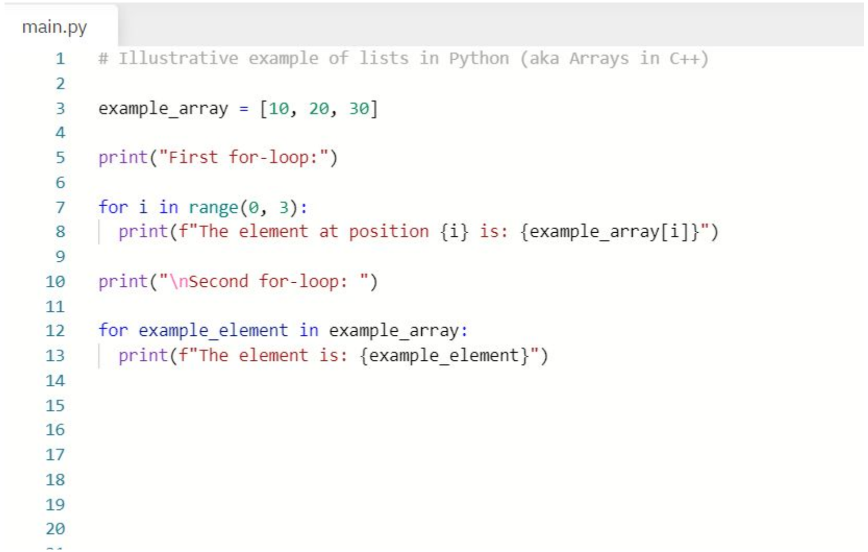
Python version using Lists



ain.cpp

```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

#include <iostream>

using namespace std;

int main()
{

  int example_array[3] = {10, 20, 30};

  cout << "First for-loop: \n";

  for (int i = 0; i < 3; i++) {
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

C++ version

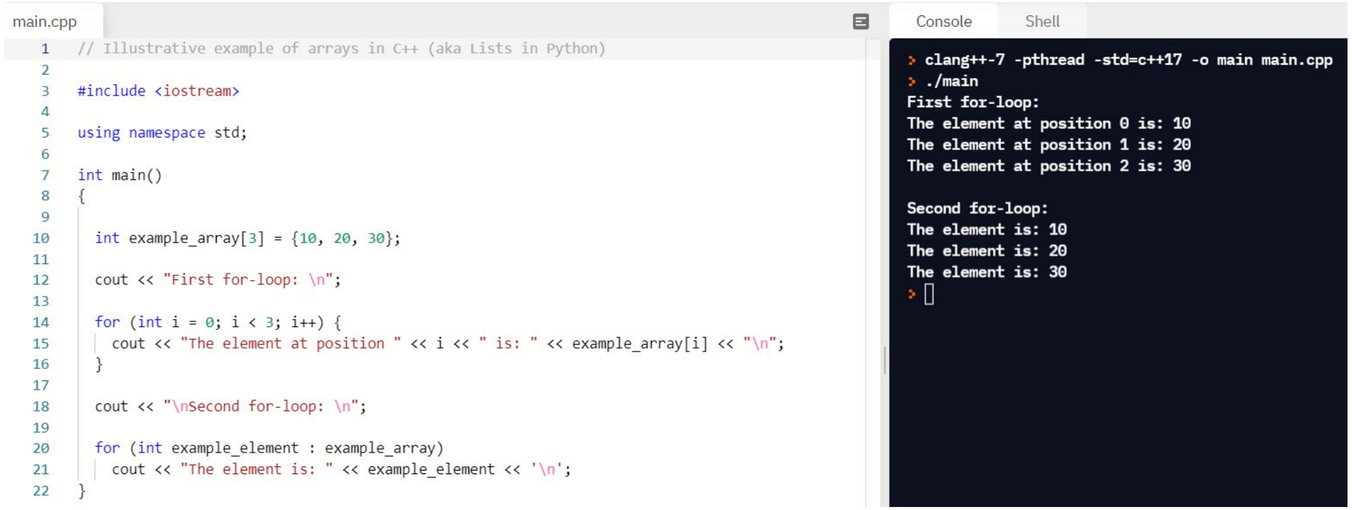


main.py

```python
# Illustrative example of lists in Python (aka Arrays in C++)

example_array = [10, 20, 30]

print("First for-loop:")

for i in range(0, 3):
  print(f"The element at position {i} is: {example_array[i]}")

print("\nSecond for-loop: ")

for example_element in example_array:
  print(f"The element is: {example_element}")
```

Python version

C++ version with Arrays:



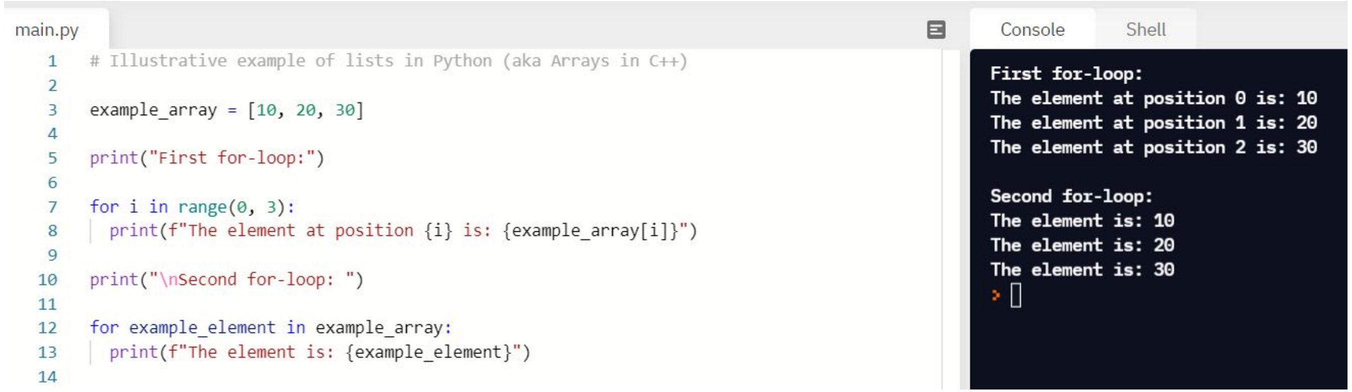
main.cpp

```cpp
// Illustrative example of arrays in C++ (aka Lists in Python)

#include <iostream>

using namespace std;

int main()
{

  int example_array[3] = {10, 20, 30};

  cout << "First for-loop: \n";

  for (int i = 0; i < 3; i++) {
    cout << "The element at position " << i << " is: " << example_array[i] << "\n";
  }

  cout << "\nSecond for-loop: \n";

  for (int example_element : example_array)
    cout << "The element is: " << example_element << '\n';
}
```

Console Shell

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

Python version with Lists:



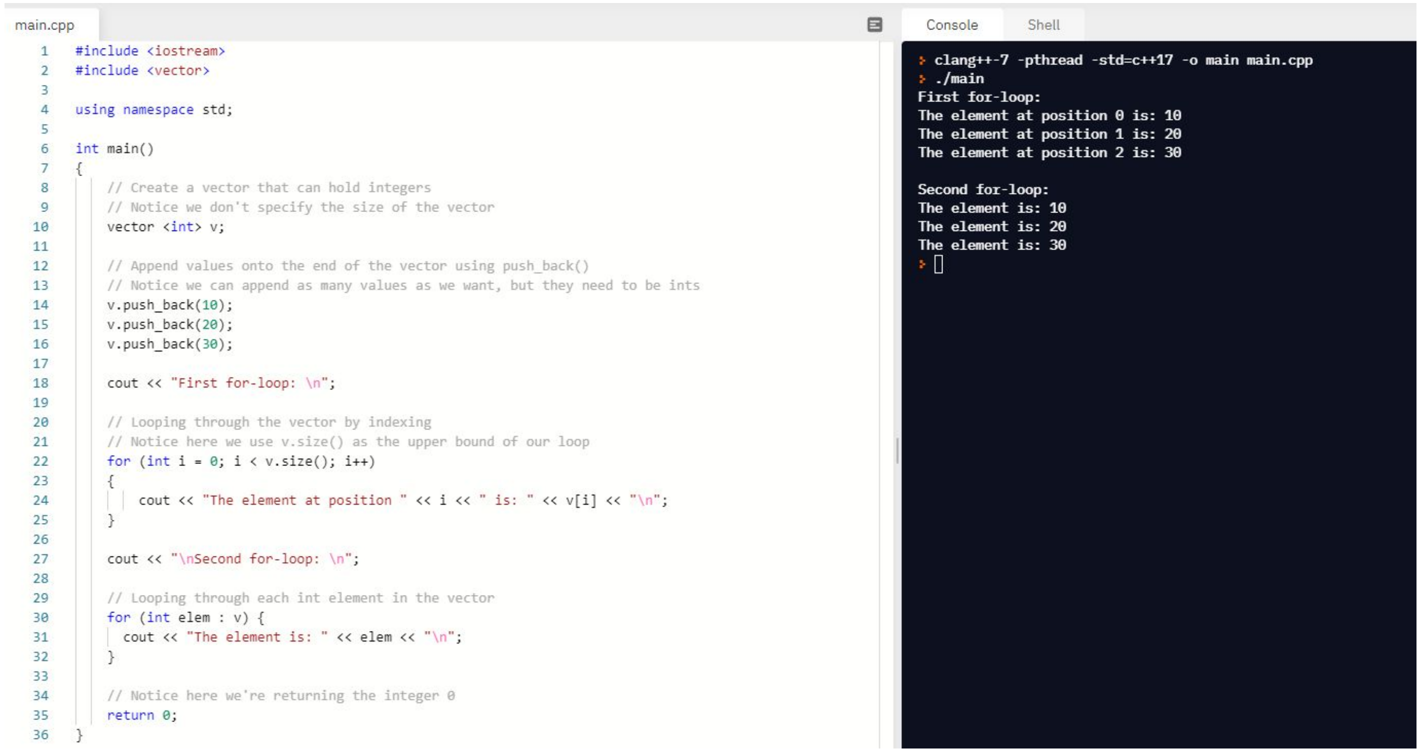
main.py

```python
# Illustrative example of lists in Python (aka Arrays in C++)

example_array = [10, 20, 30]

print("First for-loop:")

for i in range(0, 3):
  print(f"The element at position {i} is: {example_array[i]}")

print("\nSecond for-loop: ")

for example_element in example_array:
  print(f"The element is: {example_element}")

```

Console Shell

```
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
> 
```

Vectors in C++ = more like Lists in Python, still with a caveat*



main.cpp

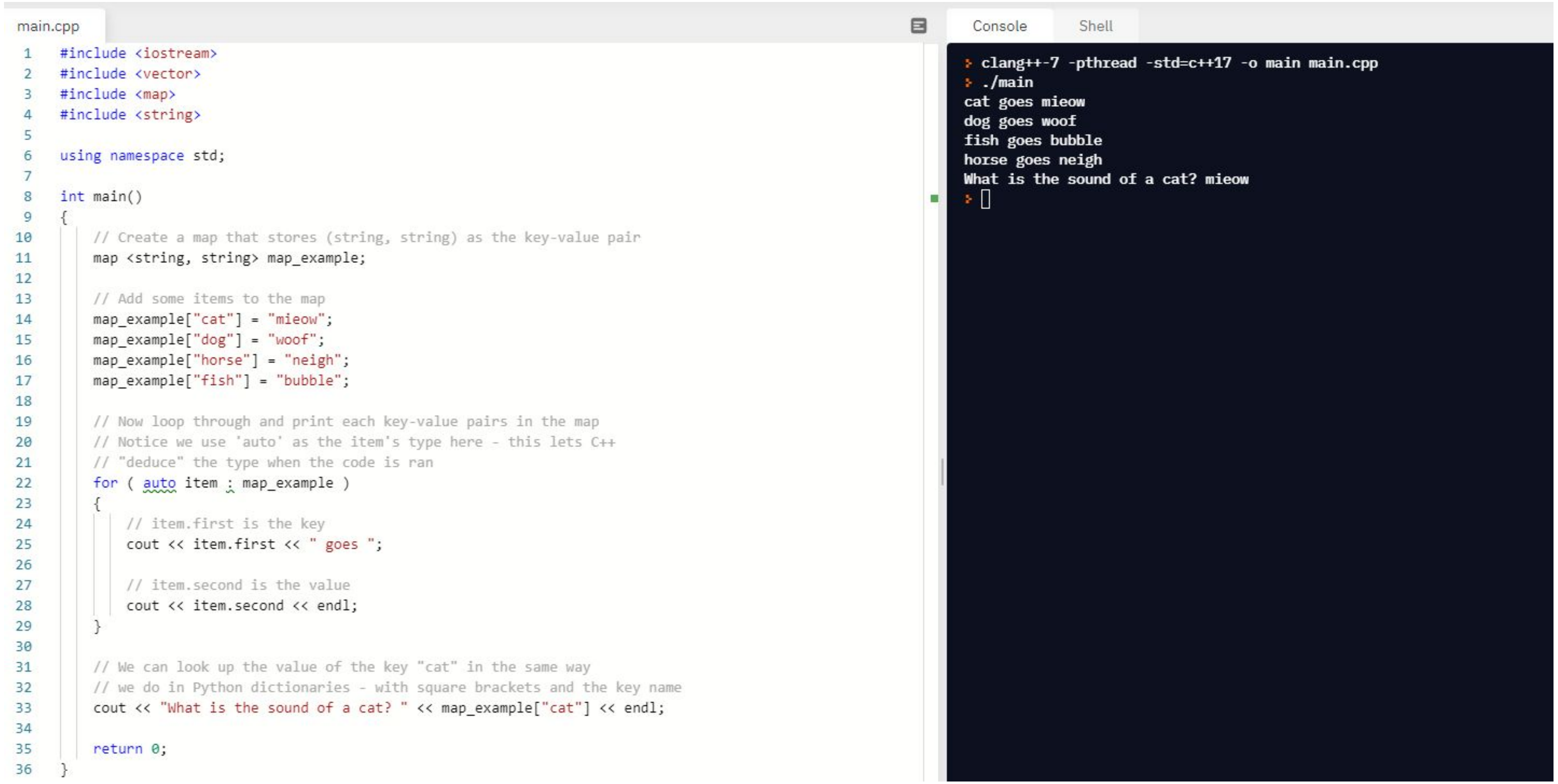
```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    // Create a vector that can hold integers
    // Notice we don't specify the size of the vector
    vector <int> v;

    // Append values onto the end of the vector using push_back()
    // Notice we can append as many values as we want, but they need to be ints
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);

    cout << "First for-loop: \n";

    // Looping through the vector by indexing
    // Notice here we use v.size() as the upper bound of our loop
    for (int i = 0; i < v.size(); i++)
    {
        cout << "The element at position " << i << " is: " << v[i] << "\n";
    }

    cout << "\nSecond for-loop: \n";

    // Looping through each int element in the vector
    for (int elem : v) {
      cout << "The element is: " << elem << "\n";
    }

    // Notice here we're returning the integer 0
    return 0;
}
```

Console   Shell

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
First for-loop:
The element at position 0 is: 10
The element at position 1 is: 20
The element at position 2 is: 30

Second for-loop:
The element is: 10
The element is: 20
The element is: 30
>
```

* When using *Vectors* in C++, you can only store values of the same type. For example, only integer or string or boolean values. However, you don't need to worry about maintaining the size of a Vector, unlike Arrays.

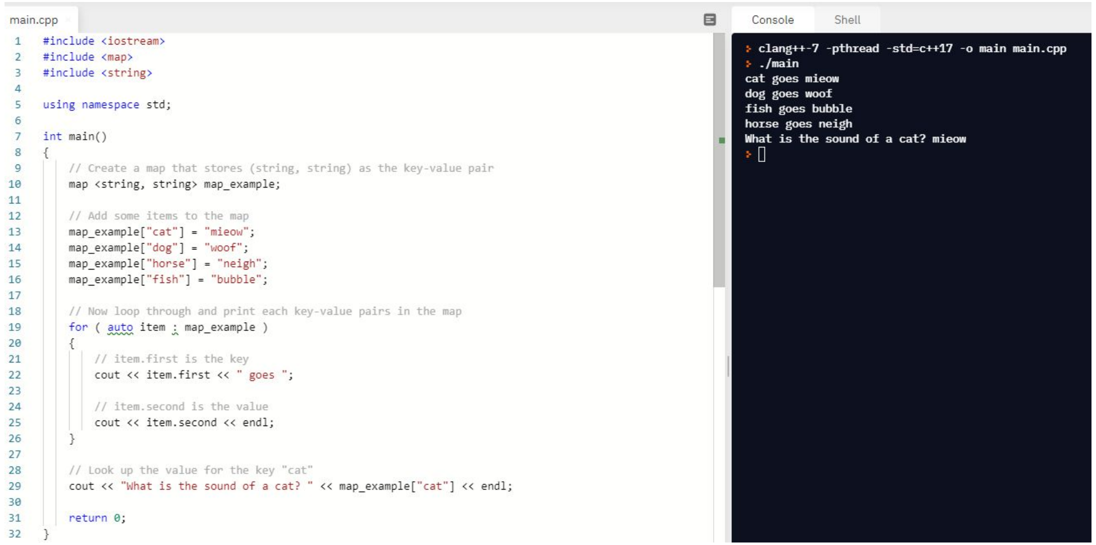
Maps in C++ = basically* Dictionaries in Python



```cpp
#include <iostream>
#include <vector>
#include <map>
#include <string>

using namespace std;

int main()
{
    // Create a map that stores (string, string) as the key-value pair
    map <string, string> map_example;

    // Add some items to the map
    map_example["cat"] = "mieow";
    map_example["dog"] = "woof";
    map_example["horse"] = "neigh";
    map_example["fish"] = "bubble";

    // Now loop through and print each key-value pairs in the map
    // Notice we use 'auto' as the item's type here - this lets C++
    // "deduce" the type when the code is ran
    for ( auto item : map_example )
    {
        // item.first is the key
        cout << item.first << " goes ";

        // item.second is the value
        cout << item.second << endl;
    }

    // We can look up the value of the key "cat" in the same way
    // we do in Python dictionaries - with square brackets and the key name
    cout << "What is the sound of a cat? " << map_example["cat"] << endl;

    return 0;
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
cat goes mieow
dog goes woof
fish goes bubble
horse goes neigh
What is the sound of a cat? mieow
>
```

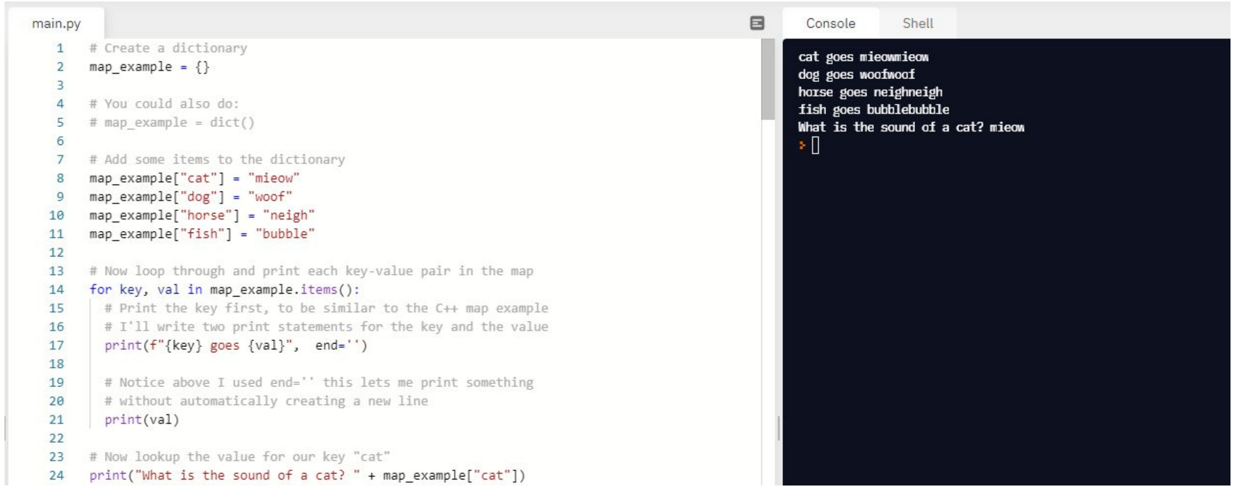
*You will need to specify the type of the *key* and the *value* in C++ when creating a map, similar to specifying the type of an array or vectors



```cpp
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main()
{
    // Create a map that stores (string, string) as the key-value pair
    map <string, string> map_example;

    // Add some items to the map
    map_example["cat"] = "mieow";
    map_example["dog"] = "woof";
    map_example["horse"] = "neigh";
    map_example["fish"] = "bubble";

    // Now loop through and print each key-value pairs in the map
    for ( auto item : map_example )
    {
        // item.first is the key
        cout << item.first << " goes ";

        // item.second is the value
        cout << item.second << endl;
    }

    // Look up the value for the key "cat"
    cout << "What is the sound of a cat? " << map_example["cat"] << endl;

    return 0;
}
```

```
> clang++-7 -pthread -std=c++17 -o main main.cpp
> ./main
cat goes mieow
dog goes woof
fish goes bubble
horse goes neigh
What is the sound of a cat? mieow
>
```

C++ version using **maps**



```python
# Create a dictionary
map_example = {}

# You could also do:
# map_example = dict()

# Add some items to the dictionary
map_example["cat"] = "mieow"
map_example["dog"] = "woof"
map_example["horse"] = "neigh"
map_example["fish"] = "bubble"

# Now loop through and print each key-value pair in the map
for key, val in map_example.items():
  # Print the key first, to be similar to the C++ map example
  # I'll write two print statements for the key and the value
  print(f"{key} goes {val}",  end='')

  # Notice above I used end='' this lets me print something
  # without automatically creating a new line
  print(val)

# Now lookup the value for our key "cat"
print("What is the sound of a cat? " + map_example["cat"])
```

```
cat goes mieowmieow
dog goes woofwoof
horse goes neighneigh
fish goes bubblebubble
What is the sound of a cat? mieow
>
```

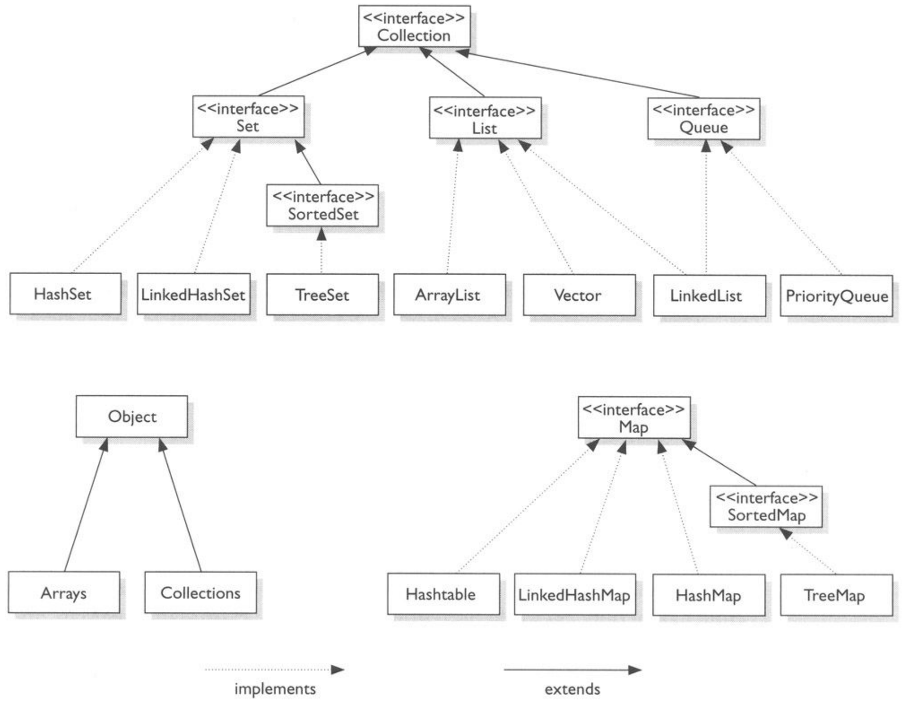
Python version using **dictionaries**

Main takeaways

- C++ is a lot pickier than Python
  - Don’t even get me started on C (*C is really powerful though*)
- When you take coding courses in college, you will likely need to implement these data collections / containers we’ve talked about (sets, dictionaries/maps, vectors/lists), from scratch in C++ without using libraries (same with algorithms!)
- Want to learn more about the differences between C++ and Python data containers? https://chryswoods.com/beginning_c++/lists.html
- (cont.)

Main takeaways

- C/C++ are really efficient compared to Python
- There are also MANY different containers for data. There is an area of CS, "data structures" which studies how we assign data memory in our computers



Modules & Libraries in Python

Modules we’ve used

- When we refer to a *module*, we are referring to a single Python file
- Examples of modules we’ve used already:
  - **Time**
    - We could put the program to “sleep” using time.sleep(x), where ‘x’ was the number of seconds we wanted the program to wait
  - **Random**
    - We could randomly pull a number between ‘x’ and ‘y’ using random.randint(x, y)
  - **Math**
    - We could round a decimal value ‘x’ up or down using math.ceil(x) or math.floor(x), respectively

Libraries

Libraries in Python, on the other hand, are a *collection* of modules.

If you're interested in more of the differences, go read this great stackoverflow* post: https://stackoverflow.com/questions/7948494/whats-the-difference-between-a-python-module-and-a-python-package

**stack overflow is a great resource / community when you're trying to figure out a bug in your code or understand a concept in programming*

Scikit-learn

Scikit-learn is an open-source Python library used for machine learning. By using advanced statistical data analysis (as in, performing some statistics on our data), machine learning can let us identify patterns to make predictions, recommendations, inferences, etc. with little human intervention.

- To get a better introduction of machine learning (“ML”), and how you can use scikit-learn to do ML, start here: https://scikit-learn.org/stable/tutorial/basic/tutorial.html
- Today’s lab will be using scikit-learn to practice a few popular ML algorithms

A fun example of machine learning: https://quickdraw.withgoogle.com/

(“This is a game built with machine learning. You draw, and a neural network tries to guess what you’re drawing. Of course, it doesn’t always work. But the more you play with it, the more it will learn. It’s just one example of how you can use machine learning in fun ways.”)

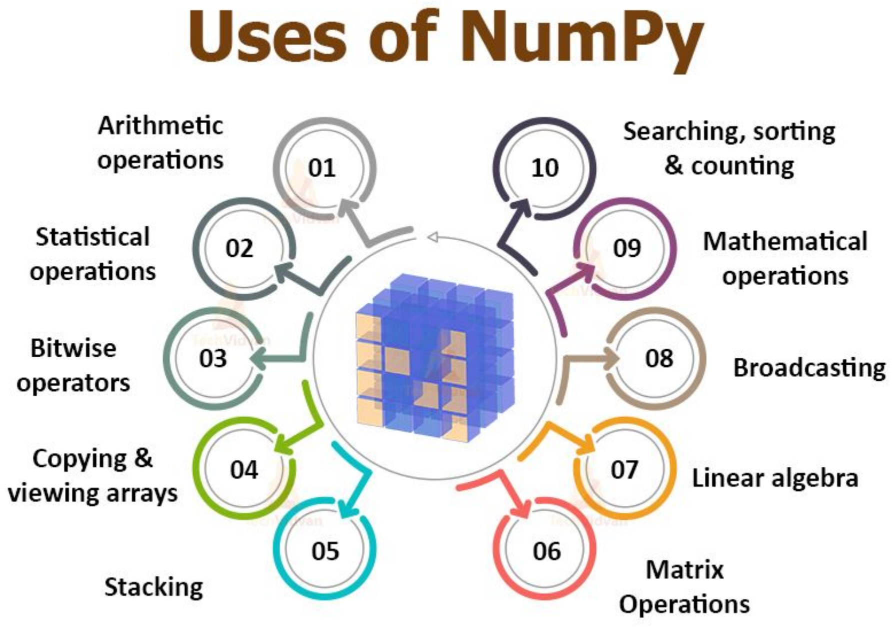
Tips for working with libraries / packages

- As you've been hearing from many of our guest speakers, **take advantage of libraries!** They are fast, helpful, and popular libraries will be well documented.
  - Plus, because they're open-source, there's a large community answering questions, debugging problems, posting guides, etc.
- Whenever you want to use a certain function from a library, read the documentation carefully! It's helpful.
  - Read about the input, or parameters, that the function expects. How many? What should the parameter's type be (e.g., int, list, something else)? Are there any 'optional' parameters? And why are they optional? (example: NumPy's arange() or matplotlib's yticks)
  - Read about the expected *output* of the function. How does the output depend on the input?
  - Find examples of people using that function on stackoverflow, reddit, whatever
    - Read the questions they post and try to understand the question BEFORE reading the answer

NumPy (Numerical Python)

NumPy is an open-source library in Python that excels mathematical, statistical, and low-level operations on complex, multi-dimensional data (like multi-dimensional arrays*). For example, if you need to do linear algebra or statistical analysis, NumPy is your go to.

*Don't be afraid of the term "multi-dimensional arrays" -- we worked with multi-dimensional lists in our 2048 board game lab! We used a list of lists to represent our board - aka a 2-dimensional list. In other words: a multidimensional array!



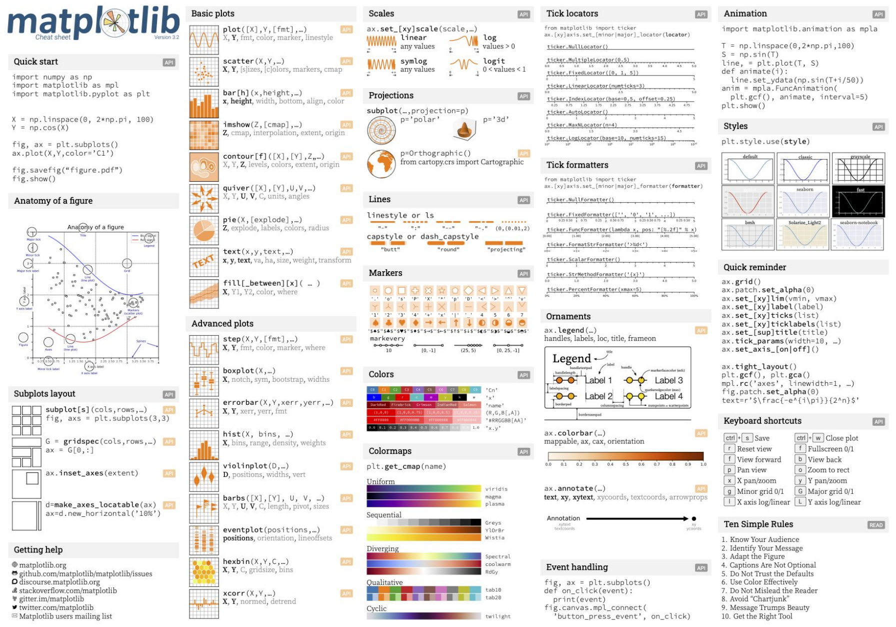
Tutorial: https://www.tutorialspoint.com/numpy/index.htm

Matplotlib

Matplotlib is a cross-platform, open-source Python library. It’s a great alternative to MATLAB. Matplotlib lets you plot or visualize data in many ways. You can create basic bar charts, scatterplots, etc. and you can also make interactive charts!

Examples of different charts:

https://matplotlib.org/stable/tutorials/introductory/sample_plots.html



matplotlib
Cheat sheet
Version 3.2

Quick start

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt


X = np.linspace(0, 2*np.pi, 100)
Y = np.cos(X)

fig, ax = plt.subplots()
ax.plot(X,Y,color='C1')

fig.savefig("figure.pdf")
fig.show()
```

Anatomy of a figure

Anatomy of a figure
Title
Major tick
Minor tick
Major tick label
Legend
Grid
Line (line plot)
Y axis label
Markers (scatter plot)
Figure
Axes
Spines
Minor tick label
X axis label

Subplots layout

```
subplot[s](cols,rows,…)
fig, axs = plt.subplots(3,3)

G = gridspec(cols,rows,…)
ax = G[0,:]

ax.inset_axes(extent)

d=make_axes_locatable(ax)
ax=d.new_horizontal('10%')
```

Getting help

- matplotlib.org
- github.com/matplotlib/matplotlib/issues
- discourse.matplotlib.org
- stackoverflow.com/matplotlib
- gitter.im/matplotlib
- twitter.com/matplotlib
- Matplotlib users mailing list

Basic plots

`plot([X],Y,[fmt],…)`	X, Y, fmt, color, marker, linestyle
`scatter(X,Y,…)`	X, Y, [s]izes, [c]olors, markers, cmap
`bar[h](x,height,…)`	x, height, width, bottom, align, color
`imshow(Z,[cmap],…)`	Z, cmap, interpolation, extent, origin
`contour[f]([X],[Y],Z,,…)`	X, Y, Z, levels, colors, extent, origin
`quiver([X],[Y],U,V,…)`	X, Y, U, V, C, units, angles
`pie(X,[explode],…)`	Z, explode, labels, colors, radius
`text(x,y,text,…)`	x, y, text, va, ha, size, weight, transform
`fill[_between][x]( … )`	X, Y1, Y2, color, where

Advanced plots

`step(X,Y,[fmt],…)`	X, Y, fmt, color, marker, where
`boxplot(X,…)`	X, notch, sym, bootstrap, widths
`errorbar(X,Y,xerr,yerr,…)`	X, Y, xerr, yerr, fmt
`hist(X, bins, …)`	X, bins, range, density, weights
`violinplot(D,…)`	D, positions, widths, vert
`barbs([X],[Y], U, V, …)`	X, Y, U, V, C, length, pivot, sizes
`eventplot(positions,…)`	positions, orientation, lineoffsets
`hexbin(X,Y,C,…)`	X, Y, C, gridsize, bins
`xcorr(X,Y,…)`	X, Y, normed, detrend

Scales

`ax.set_[xy]scale(scale,…)`

linear	any values	log	values > 0
symlog	any values	logit	0 < values < 1

Projections

`subplot(…,projection=p)`

p='polar' p='3d'

p=Orthographic()
from cartopy.crs import Cartographic

Lines

linestyle or ls

"-" ":" "--" "-." (0,(0.01,2))

capstyle or dash_capstyle

"butt" "round" "projecting"

Markers

'.' 'o' 's' 'P' 'X' '*' 'p' 'D' '<' '>' '^' 'v'
'1' '2' '3' '4' '+' 'x' '|' '_' 4 5 6 7
'$♠$' "$♣$" "$♥$" "$♦$" "$→$" "$←$" "$↑$" "$↓$" "$◐$" "$◑$" "$◒$" "$◓$'

markevery

10 [0, -1] (25, 5) [0, 25, -1]

Colors

C0 C1 C2 C3 C4 C5 C6 C7 C8 C9	'Cn'
b g r c m y k w	'x'
DarkRed Firebrick Crimson IndianRed Salmon	'name'
(1,0,0) (1,0,0,0.75) (1,0,0,0.5) (1,0,0,0.25)	(R,G,B[,A])
#FF0000 #FF0000BB #FF000088 #FF000044	'#RRGGBB[AA]'
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0	'x.y'

Colormaps

`plt.get_cmap(name)`

Uniform: viridis, magma, plasma
Sequential: Greys, YlOrBr, Wistia
Diverging: Spectral, coolwarm, RdGy
Qualitative: tab10, tab20
Cyclic: twilight

Tick locators

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_locator(locator)
```

```
ticker.NullLocator()
ticker.MultipleLocator(0.5)
ticker.FixedLocator([0, 1, 5])
ticker.LinearLocator(numticks=3)
ticker.IndexLocator(base=0.5, offset=0.25)
ticker.AutoLocator()
ticker.MaxNLocator(n=4)
ticker.LogLocator(base=10, numticks=15)
```

Tick formatters

```
from matplotlib import ticker
ax.[xy]axis.set_[minor|major]_formatter(formatter)
```

```
ticker.NullFormatter()
ticker.FixedFormatter(['', '0', '1', ...])
ticker.FuncFormatter(lambda x, pos: "[%.2f]" % x)
ticker.FormatStrFormatter('>%d<')
ticker.ScalarFormatter()
ticker.StrMethodFormatter('{x}')
ticker.PercentFormatter(xmax=5)
```

Ornaments

`ax.legend(…)`
handles, labels, loc, title, frameon

`ax.colorbar(…)`
mappable, ax, cax, orientation

`ax.annotate(…)`
text, xy, xytext, xycoords, textcoords, arrowprops

Event handling

```
fig, ax = plt.subplots()
def on_click(event):
  print(event)
fig.canvas.mpl_connect(
  'button_press_event', on_click)
```

Animation

```
import matplotlib.animation as mpla

T = np.linspace(0,2*np.pi,100)
S = np.sin(T)
line, = plt.plot(T, S)
def animate(i):
  line.set_ydata(np.sin(T+i/50))
anim = mpla.FuncAnimation(
  plt.gcf(), animate, interval=5)
plt.show()
```

Styles

`plt.style.use(style)`

Quick reminder

```
ax.grid()
ax.patch.set_alpha(0)
ax.set_[xy]lim(vmin, vmax)
ax.set_[xy]label(label)
ax.set_[xy]ticks(list)
ax.set_[xy]ticklabels(list)
ax.set_[sup]title(title)
ax.tick_params(width=10, …)
ax.set_axis_[on|off]()

ax.tight_layout()
plt.gcf(), plt.gca()
mpl.rc('axes', linewidth=1, …)
fig.patch.set_alpha(0)
text=r'$\frac{-e^{i\pi}}{2^n}$'
```

Keyboard shortcuts

ctrl + s	Save	ctrl + w	Close plot
r	Reset view	f	Fullscreen 0/1
f	View forward	b	View back
p	Pan view	o	Zoom to rect
x	X pan/zoom	y	Y pan/zoom
g	Minor grid 0/1	G	Major grid 0/1
l	X axis log/linear	L	Y axis log/linear

Ten Simple Rules READ

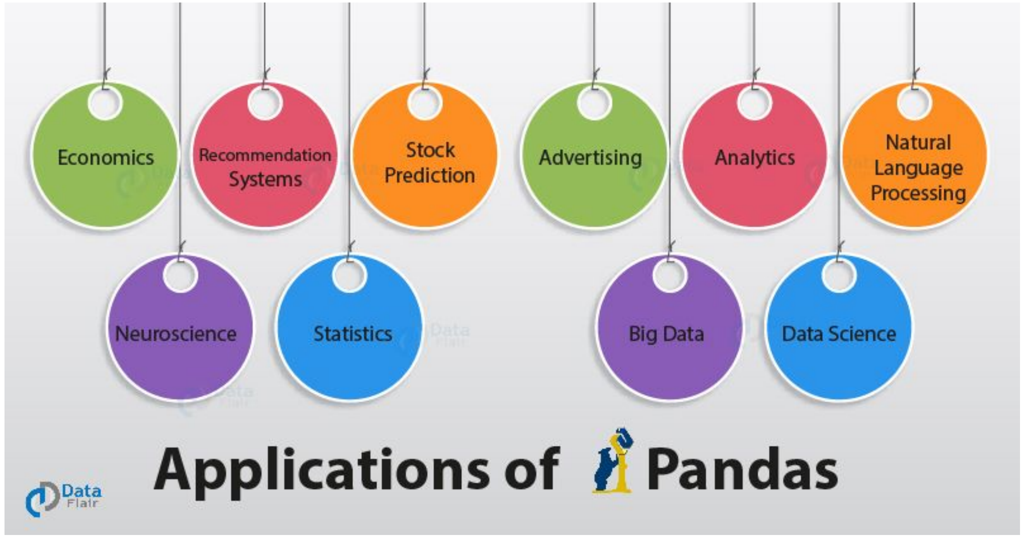
1. Know Your Audience
2. Identify Your Message
3. Adapt the Figure
4. Captions Are Not Optional
5. Do Not Trust the Defaults
6. Use Color Effectively
7. Do Not Mislead the Reader
8. Avoid "Chartjunk"
9. Message Trumps Beauty
10. Get the Right Tool

Cheatsheet:
https://raw.githubusercontent.com/matplotlib/cheatsheets/master/cheatsheets-1.png

Pandas

Pandas is an open-source Python library used for data analysis and data science, its especially useful for reading from files or tables.

It's great when you're working with large amounts of data you want to analyze: like social media trends, or trying to predicting the price of the stock market.



Tutorial to get started: https://www.w3schools.com/python/pandas/default.asp

No live coding today!

If we finish early, go start reading the documentation for scikit-learn’s decision trees: you’ll be using some of the functions in today’s lab. Alternatively, go to Lecture-07 from yesterday and try building out the rest of the shopping cart program using a dictionary.