

# CS229 Section: Python Tutorial

**Maya Srikanth**

Content adapted from past CS229 iterations

# Python

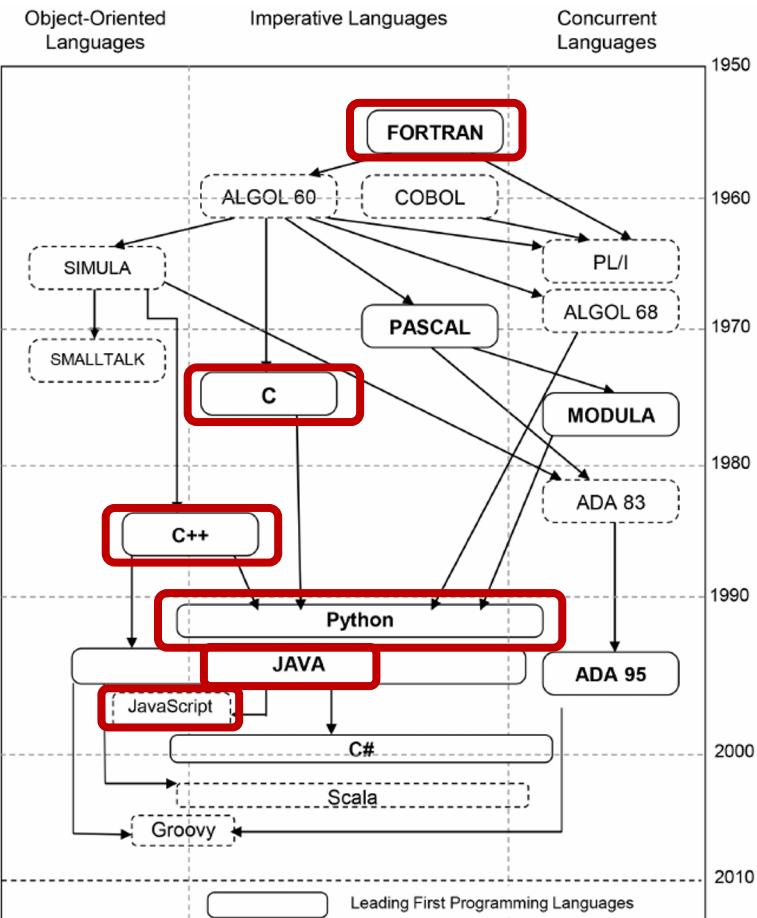
Python 2.0 released in 2000

(Python 2.7 “end-of-life” in  
2020)

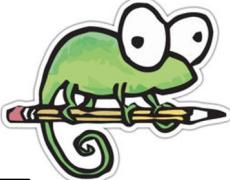
Python 3.0 released in 2008

(Python 3.6+ for CS 229)

- High-level object-oriented, interpreted  
language



# Text editor/IDE options.. (don't settle with notepad)

- PyCharm (IDE) 
- Visual Studio Code (IDE) 
- Sublime Text (IDE) 
- Atom 
- Notepad ++/gedit 
- Vim (for Linux) 



# PyCharm IDE

## PyCharm

- Good debugger
- Project management

The screenshot shows the PyCharm IDE interface. The code editor displays a Python script named `script.py` with several imports and functions. A red dot marks the current line of execution. The bottom right shows the 'Evaluate Expression' dialog with the variable `positive` selected, and its value is shown in the 'Result' panel.

```
27     """A map from strings to their counts in 'tree'. (Counter maps a
28     list to a dict of counts of the elements in that list.)
29
30     """
31     return Counter(tree.leaves())
32
33     from nltk.corpus import opinion_lexicon
34
35     # Use set for fast membership checking:
36     positive = set(opinion_lexicon.positive())
37     negative = set(opinion_lexicon.negative())
38
39     def op_ungigrams_phi(tree):
40         words = tree.leaves()
41         filtered = [w for w in words if w in positive or w in negative]
42         return Counter(filtered)
43
44     def fit_Si(mod):
45         mod = f
46         S_i = mod.m
47         mod.f = f
48         return mod
49
50     result = (set('kindly', 'well-balanced', 'buoyant', 'dumbfounding', 'cheerful', 'excellent', 'graceful', 'thoughtful', 'gained', 'mesmerize', 'faultless', 'well-reasoned', 'admirer', 'acclaim', 'acumen', 'agilely', 'angel', 'aspire', 'appealing', 'abundant', 'accessible', 'user-replaceable'))
```

Evaluate Expression  
positive

Result:

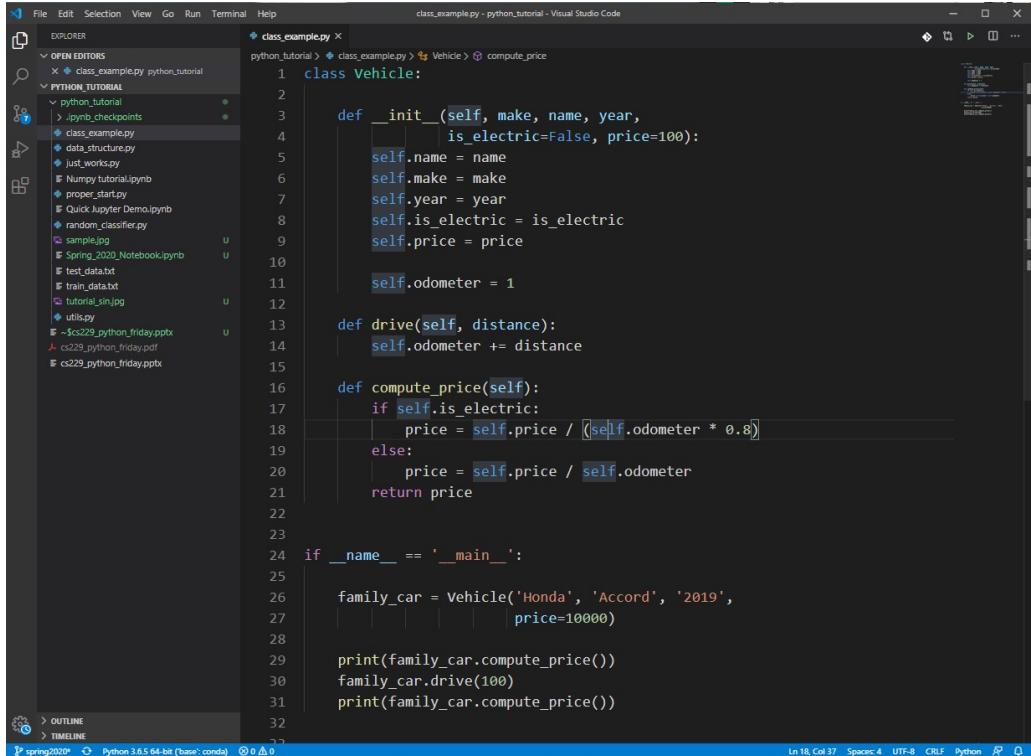
- result = (set('kindly', 'well-balanced', 'buoyant', 'dumbfounding', 'cheerful', 'excellent', 'graceful', 'thoughtful', 'gained', 'mesmerize', 'faultless', 'well-reasoned', 'admirer', 'acclaim', 'acumen', 'agilely', 'angel', 'aspire', 'appealing', 'abundant', 'accessible', 'user-replaceable'))
- len\_ = (int) 2006
- 112746468160 = (str) 'admirer'
- 112746468216 = (str) 'acclaim'
- 112746468832 = (str) 'acumen'
- 112746470064 = (str) 'agilely'
- 112746470960 = (str) 'angel'
- 112746471352 = (str) 'aspire'
- 112746492784 = (str) 'appealing'
- 112746494768 = (str) 'abundant'
- 112746495856 = (str) 'accessible'
- 112777249896 = (str) 'user-replaceable'

FYI, professional version free for students: <https://www.jetbrains.com/student/>

# Visual Studio IDE

## Visual Studio Code

- Light weight
- Wide variety of plugins to enable support for all languages



A screenshot of the Visual Studio Code interface. The left sidebar shows a file tree with several Python files and other files like images and PDFs. The main editor area displays a Python script named `class_example.py`. The code defines a `Vehicle` class with methods for initialization, driving, and computing price based on whether it's electric or not. It also includes a check for the `__name__` variable and prints the price of a family car.

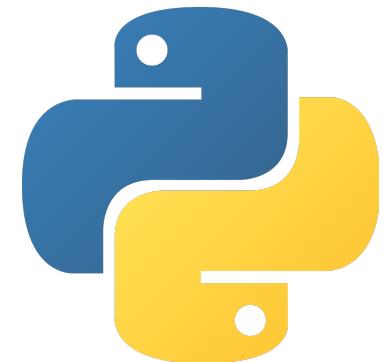
```
class Vehicle:
    def __init__(self, make, name, year, is_electric=False, price=100):
        self.name = name
        self.make = make
        self.year = year
        self.is_electric = is_electric
        self.price = price

    def drive(self, distance):
        self.odometer += distance

    def compute_price(self):
        if self.is_electric:
            price = self.price / (self.odometer * 0.8)
        else:
            price = self.price / self.odometer
        return price

if __name__ == '__main__':
    family_car = Vehicle('Honda', 'Accord', '2019',
                         price=10000)
    print(family_car.compute_price())
    family_car.drive(100)
    print(family_car.compute_price())
```

# Basic Python: Strings, Lists, Dictionaries



# String manipulation

## Formatting

```
print('I love CS229. (upper)'.upper())
print('I love CS229. (rjust 20)'.rjust(20))
print('we love CS229. (capitalize)'.capitalize())
print('      I love CS229. (strip)      '.strip())
```

## Concatenation

```
print('I like ' + str(cs_class_code) + ' a lot!')
print(f'{print} (print a function)')
print(f'{type(229)} (print a type)')
```

## Formatting

```
print('Old school formatting: {:.2F}'.format(1.358))
```

# List

```
list_1 = ['one', 'two', 'three']
```

## List creation

```
list_1.append(4)  
list_1.insert(0, 'ZERO')
```

## Insertion/extension

```
list_2 = [1, 2, 3]  
list_1.extend(list_2)
```

## List comprehension

```
long_list = [i for i in range(9)]  
long_long_list = [(i, j) for i in range(3)  
                  for j in range(5)]  
long_list_list = [[i for i in range(3)]  
                  for _ in range(5)]
```

## Sorting

```
sorted(random_list)  
random_list_2 = [(3, 'z'), (12, 'r'), (6, 'e'),  
                 (8, 'c'), (2, 'g')]  
sorted(random_list_2, key=lambda x: x[1])
```

# Dictionary and Set

Set

(unordered, unique)

```
my_set = {i ** 2 for i in range(10)}  
{0, 1, 64, 4, 36, 9, 16, 49, 81, 25}
```

Dictionary  
(mapping)

```
my_dict = {(5-i): i ** 2 for i in range(10)}  
{5: 0, 4: 1, 3: 4, 2: 9, 1: 16, 0: 25, -1:  
36, -2: 49, -3: 64, -4: 81}  
dict_keys([5, 4, 3, 2, 1, 0, -1, -2, -3, -  
4])
```

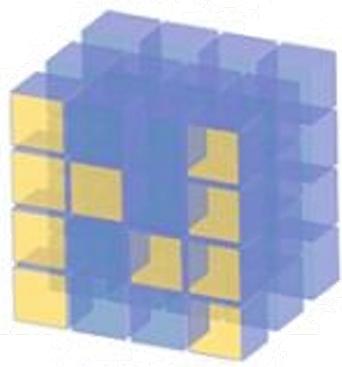
Dictionary update

```
second_dict = {'a': 10, 'b': 11}  
my_dict.update(second_dict)
```

Iterate through items

```
for k, it in my_dict.items():  
    print(k, it)
```

# NumPy



# What is NumPy and why?

- Package for scientific computing in Python
  - Vector and matrix manipulation
  - Broadcasting and vectorization (matrix operations)  
saves time & cleans up code

# Convenient math functions, read before use!

Python Command	Description
np.linalg.inv	Inverse of matrix (numpy as equivalent)
np.linalg.eig	Get eigen values & eigen vectors of arr
np.matmul	Matrix multiply
np.zeros	Create a matrix filled with zeros (Read on np.ones)
np.arange	Start, stop, step size (Read on np.linspace)
np.identity	Create an identity matrix
np.vstack	Vertically stack 2 arrays (Read on np.hstack)

# Debugging tools...

Python Command	Description
array.shape	Get shape of numpy array
array.dtype	Check data type of array (for precision, for weird behavior)
type(stuff)	Get type of a variable
import pdb; pdb.set_trace()	Set a breakpoint ( <a href="https://docs.python.org/3/library/pdb.html">https://docs.python.org/3/library/pdb.html</a> )
print(f'My name is {name}')	Easy way to construct a message

# Basic NumPy Usage

## Initialization from Python lists

```
array_1d = np.array([1, 2, 3, 4])
array_1by4 = np.array([[1, 2, 3, 4]])
large_array = np.array([i for i in range(400)])
large_array = large_array.reshape((20, 20))
```

Lists with different types  
(NumPy auto-casts to higher precision, but it should be reasonably consistent)

```
from_list = np.array([1, 2, 3])
from_list_2d = np.array([[1, 2, 3.0], [4, 5, 6]])
from_list_bad_type = np.array([1, 2, 3, 'a'])

print(f'Data type of integer is {from_list.dtype}')
print(f'Data type of float is {from_list_2d.dtype}')
```

NumPy supports many types of algebra on an entire array

```
array_1 + 5
array_1 * 5
np.sqrt(array_1)
np.power(array_1, 2)
np.exp(array_1)
np.log(array_1)
```

# Dot product and matrix multiplication

A few ways to write dot product

```
array_1 @ array_2  
array_1.dot(array_2)  
np.dot(array_1, array_2)
```

Matrix multiplication like Ax

```
weight_matrix = np.array([1, 2, 3, 4]).reshape(2, 2)  
sample = np.array([[50, 60]]).T  
np.matmul(weight_matrix, sample)
```

2D matrix multiplication

```
mat1 = np.array([[1, 2], [3, 4]])  
mat2 = np.array([[5, 6], [7, 8]])  
np.matmul(mat1, mat2)
```

Element-wise multiplication

```
a = np.array([i for i in range(10)]).reshape(2, 5)  
a * a  
np.multiply(a, a)  
np.multiply(a, 10)
```

# Broadcasting

NumPy compares dimensions of operands, then infers missing/mismatched dimensions so the operation is still valid. Be careful with dimensions!

```
op1 = np.array([i for i in range(9)]).reshape(3, 3)
op2 = np.array([[1, 2, 3]])
op3 = np.array([1, 2, 3])
```

```
# Notice that the results here are DIFFERENT!
print(op1 + op2)
print(op1 + op2.T)
```

```
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11]])
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11]])
```

---

```
# Notice that the results here are THE SAME!
print(op1 + op3)
print(op1 + op3.T)
```

```
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11]])
array([[ 1,  3,  5],
       [ 4,  6,  8],
       [ 7,  9, 11]])
```

# Broadcasting for pairwise distance

```
samples = np.random.random((15, 5))

# Without broadcasting
expanded1 = np.expand_dims(samples, axis=1)
tile1 = np.tile(expanded1, (1, samples.shape[0], 1))
expanded2 = np.expand_dims(samples, axis=0)
tile2 = np.tile(expanded2, (samples.shape[0], 1, 1))
diff = tile2 - tile1
distances = np.linalg.norm(diff, axis=-1)
```

```
# With broadcasting
diff = samples[:, np.newaxis, :]
          - samples[np.newaxis, :, :]
distances = np.linalg.norm(diff, axis=-1)
```

```
# With scipy (another math toolbox)
import scipy
distances = scipy.spatial.distance.cdist(samples, samples)
```

Both achieve the effect

$$\begin{bmatrix} \vec{a} \\ \vec{b} \end{bmatrix} \Rightarrow \begin{bmatrix} \vec{a} & \vec{a} \\ \vec{b} & \vec{b} \end{bmatrix} - \begin{bmatrix} \vec{a} & \vec{b} \\ \vec{b} & \vec{b} \end{bmatrix}$$

# Why should I vectorize my code? (dot product)

Shorter code, faster execution

```
a = np.random.random(500000)  
b = np.random.random(500000)
```

With loop

```
dot = 0.0  
for i in range(len(a)):  
    dot += a[i] * b[i]
```

```
print(dot)
```

Wall time: **345ms**

Numpy dot product

```
print(np.array(a).dot(np.array(b)))
```

Wall time: **2.9ms**

# An example with pairwise distance

Speed up depends on setup and nature of computation

```
samples = np.random.random((100, 5))

With loop                                Numpy with broadcasting

total_dist = []
for s1 in samples:
    for s2 in samples:
        d = np.linalg.norm(s1 - s2)
        total_dist.append(d)

avg_dist = np.mean(total_dist)

diff = samples[:, np.newaxis, :] -          samples[np.newaxis, :, :]
distances = np.linalg.norm(diff, axis=-1)
avg_dist = np.mean(distances)
```

Wall time: **162ms**

(even worse without NumPy  
norm)

Wall time: **3.5ms**

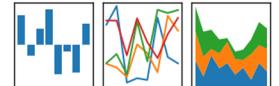


# Tools for Plotting



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



# Other Python packages/tools

## Jupyter Notebook

- Interactive, re-execution, result storage



## Matplotlib / Seaborn

- Visualization (line, scatter, bar, images and even interactive 3D)

## Pandas (<https://pandas.pydata.org/>)

- DataFrame (database/Excel-like)
- Easy filtering, aggregation (also plotting, but less features than dedicated datavis packages)

# Example plots

<https://matplotlib.org/3.1.1/gallery/index.html>

## Import

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

## Create data

```
# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)
```

## Plotting

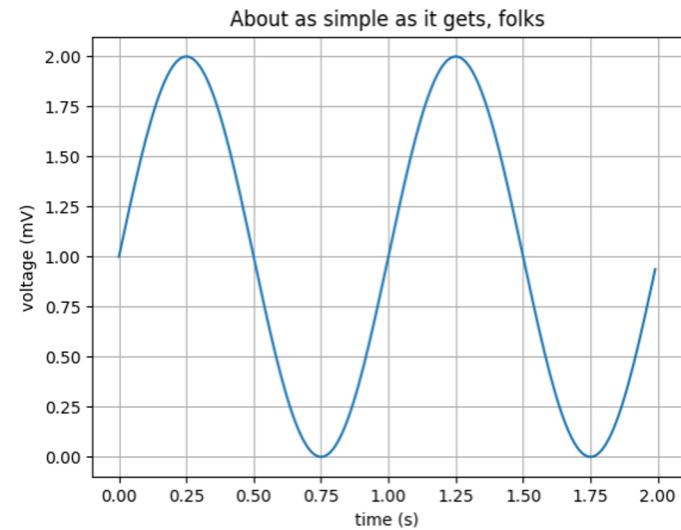
```
fig, ax = plt.subplots()
ax.plot(t, s)
```

## Format plot

```
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()
```

## Save/show

```
fig.savefig("test.png")
plt.show()
```



# Plot with dash lines and legend

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

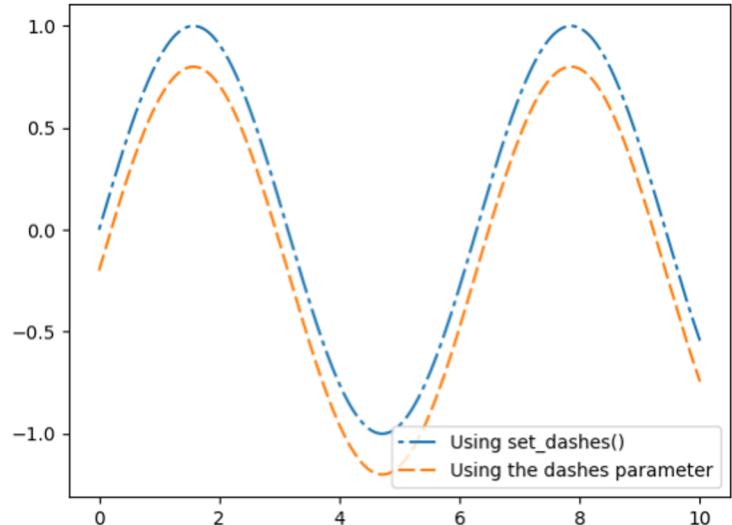
x = np.linspace(0, 10, 500)
y = np.sin(x)

fig, ax = plt.subplots()

line1, = ax.plot(x, y, label='Using set_dashes()')
# 2pt line, 2pt break, 10pt line, 2pt break
line1.set_dashes([2, 2, 10, 2])

line2, = ax.plot(x, y - 0.2, dashes=[6, 2],
                  label='Using the dashes parameter')

ax.legend()
plt.show()
```



# Using subplot

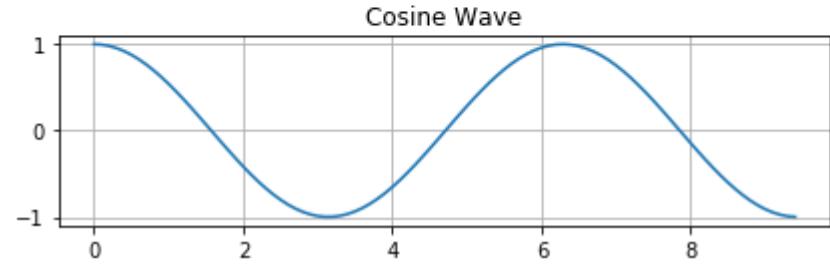
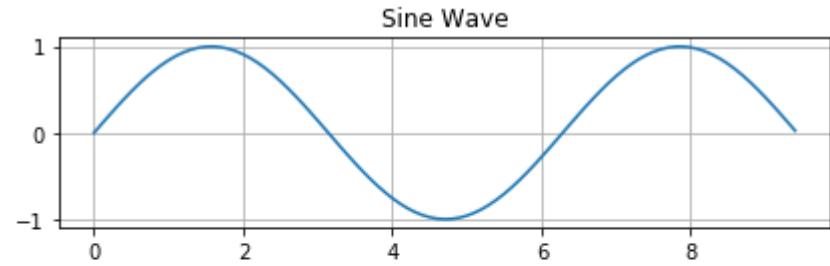
```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Setup grid with height 2 and col 1.
# Plot the 1st subplot
plt.subplot(2, 1, 1)

plt.grid()
plt.plot(x, y_sin)
plt.title('Sine Wave')

# Now plot on the 2nd subplot
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine Wave')

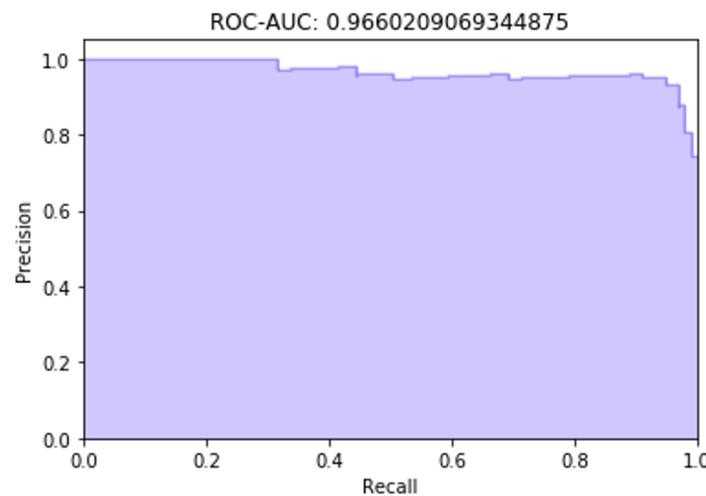
plt.grid()
plt.tight_layout()
```



# Plot area under curve

```
def prec_rec_curve(model, X, Y_true, title="", verbose=False):
    probas_pred = model.predict_proba(X)[:, 1]
    pos_label = 1.0
    precision, recall, thresholds = precision_recall_curve(Y_true,
                                                             probas_pred,
                                                             pos_label=pos_label)
    step_kwargs = ({'step': 'post'}
                  if 'step' in signature(plt.fill_between).parameters
                  else {})
    plt.step(recall, precision, color='b', alpha=0.2,
             where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title(title+ "ROC-AUC: {}".format(auc(recall, precision)))
    plt.show()
```



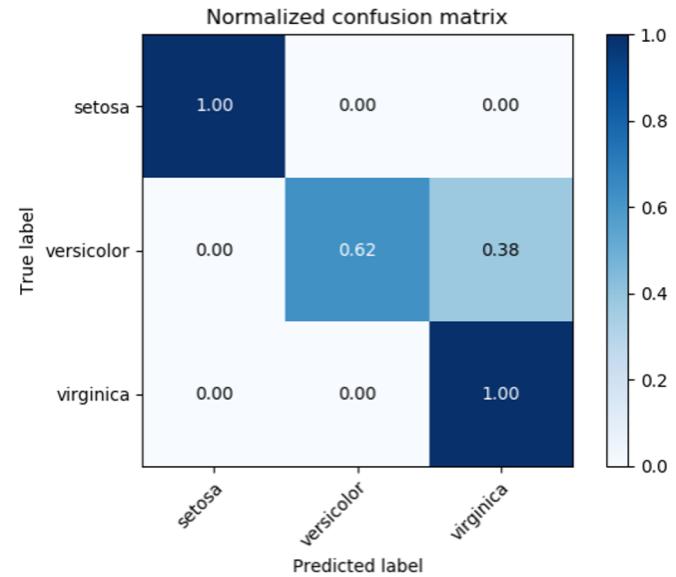
# Confusion matrix

[https://scikit-learn.org/stable/auto\\_examples/model\\_selection/plot\\_confusion\\_matrix.html](https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)

```
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=classes, yticklabels=classes,
       ylabel='True label', xlabel='Predicted label',
       title=title)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha='right',
         rotation_mode='anchor')

# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha='center', va='center',
                color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
```



- DEMO...

Good luck on HW and projects!

Questions?

# Supplementary Slides

# Where does my program start?

It just works

```
def do_something(number):
    for i in number:
        print(f'Hello {i}')

do_something(5)
```

A function

Properly

```
def do_something(number):
    for i in number:
        print(f'Hello {i}')

if __name__ == '__main__':
    do_something(5)
```

# What is a class?

Initialize the class to get an **instance** using some parameters

**Instance** variable

Does something with the **instance**

```
class Vehicle:  
    def __init__(self, make, name, year,  
                 is_electric=False, price=100):  
        self.name = name  
        self.make = make  
        self.year = year  
        self.is_electric = is_electric  
        self.price = price  
  
        self.odometer = 0  
  
    def drive(self, distance):  
        self.odometer += distance  
  
    def compute_price(self):  
        if self.is_electric:  
            price = self.price / (self.odometer * 0.8)  
        else:  
            price = self.price / self.odometer  
        return price
```

# To use a class

Instantiate a class,

get an **instance**

Call an instance method

```
if __name__ == '__main__':
    family_car = Vehicle('Honda', 'Accord', '2019',
                           price=10000)
    print(family_car.compute_price())
    family_car.drive(100)
    print(family_car.compute_price())
```

# String manipulation

## Formatting

```
stripped = '    I love CS229! '.strip()
```

```
upper_case = 'i love cs 229! '.upper()
```

```
capitalized = 'i love cs 229! '.capitalize()
```

## Concatenation

```
joined = 'string 1' + ' ' + 'string 2'
```

## Formatting

```
formatted = 'Formatted number {:.2F}'.format(1.2345)
```

# Basic data structures

## List

```
example_list = [1, 2, '3', 'four']
```

## Set (unordered, unique)

```
example_set = set([1, 2, '3', 'four'])
```

## Dictionary (mapping)

```
example_dictionary =  
    {  
        '1': 'one',  
        '2': 'two',  
        '3': 'three'  
    }
```

# More on List

## 2D list

```
list_of_list = [[1,2,3], [4,5,6], [7,8,9]]
```

## List comprehension

```
initialize_a_list = [i for i in range(9)]
```

```
initialize_a_list = [i ** 2 for i in range(9)]
```

```
initialize_2d_list = [[i + j for i in range(5)] for j in range(9)]
```

## Insert/Pop

```
my_list.insert(0, 'stuff')
```

```
print(my_list.pop(0))
```

# More on List

Sort a list

```
random_list = [3,12,5,6]
```

```
sorted_list = sorted(random_list)
```

```
random_list = [(3, 'A'),(12, 'D'),(5, 'M'),(6, 'B')]
```

```
sorted_list = sorted(random_list, key=lambda x: x[1])
```

# More on Dict/Set

## Comprehension

```
my_dict = {i: i ** 2 for i in range(10)}
```

```
my_set = {i ** 2 for i in range(10)}
```

## Get dictionary keys

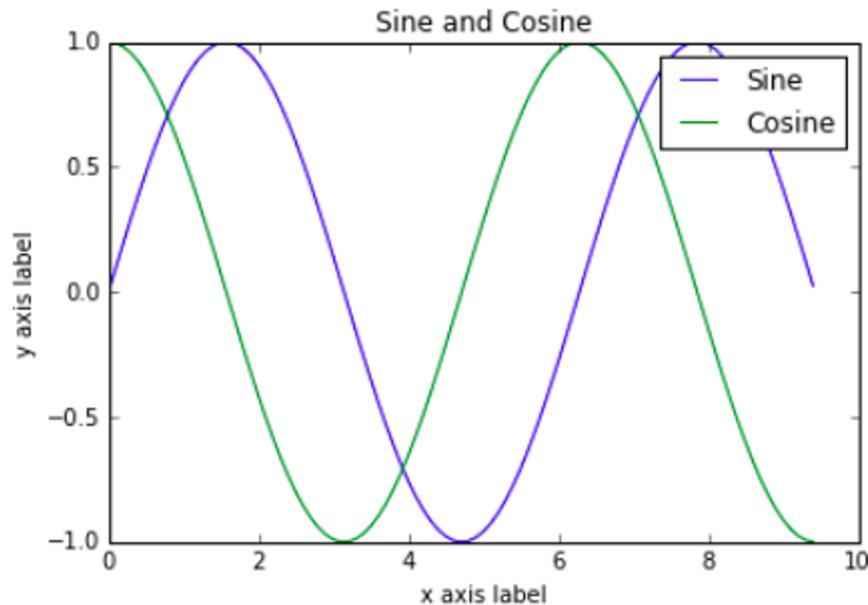
```
my_dict.keys()
```

# Another way for legend

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

# Compute the x and y coordinates for points on both curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



# Scatter plot

```
import matplotlib.pyplot as plt
import pandas as pd

girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(grades_range, girls_grades, color='r')
plt.scatter(grades_range, boys_grades, color='g')
plt.xlabel('Grades Range')
plt.ylabel('Grades Scored')
plt.show()
```

