CS 260 – Foundations of Data Science

Lecture 1 – Course Introduction

09/01/2026

Adam Poliak


Data Science is the study of extracting value from data
JEANNETTE WING

What is Data Science?

*"Data science is the study of extracting **value** from data"*

- Jeannette Wing

Value

- Requires domain expertise to determine what value is
- *Value from data* is different based on the domain and the needs

What is Data Science?

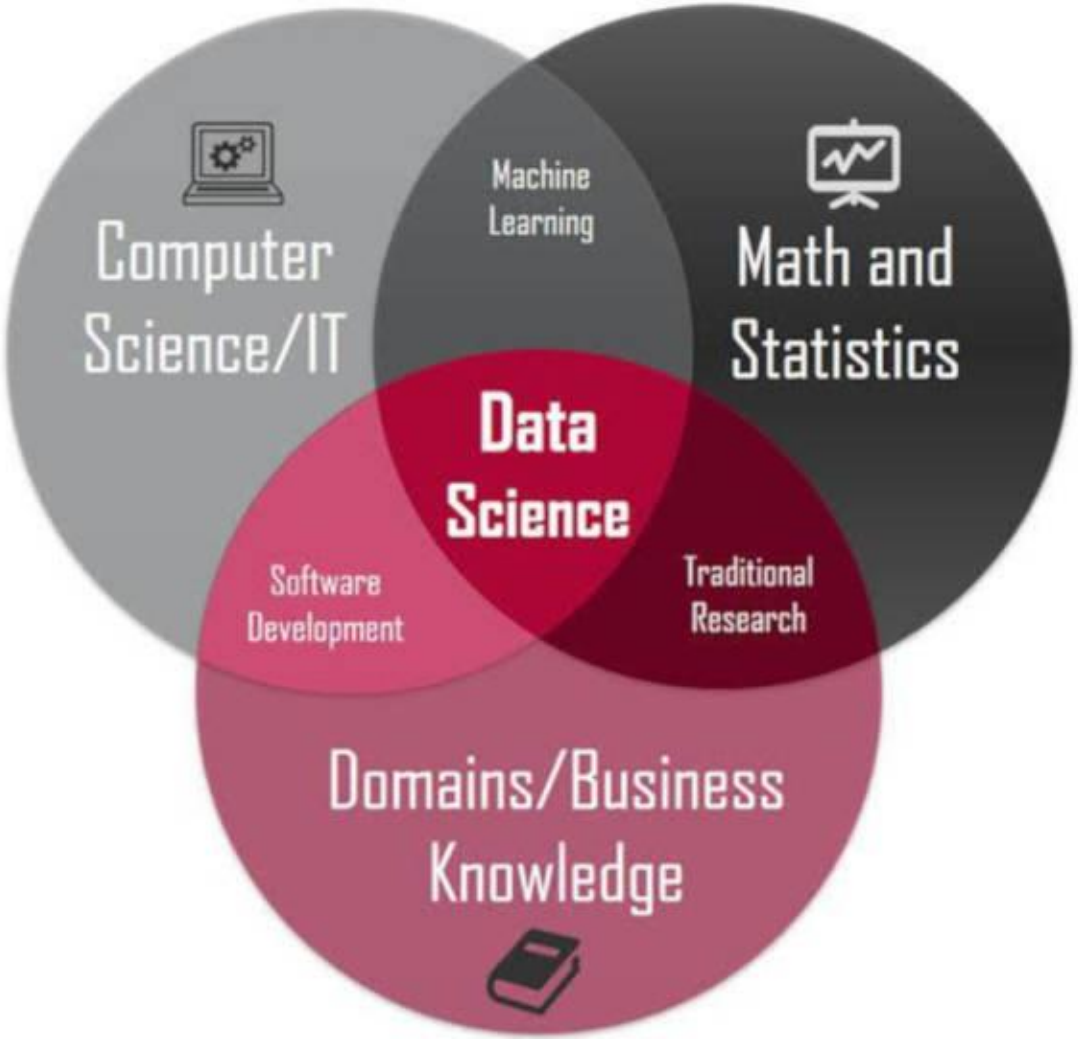
*"Data science is the study of **extracting** value from data"*

- Jeannette Wing

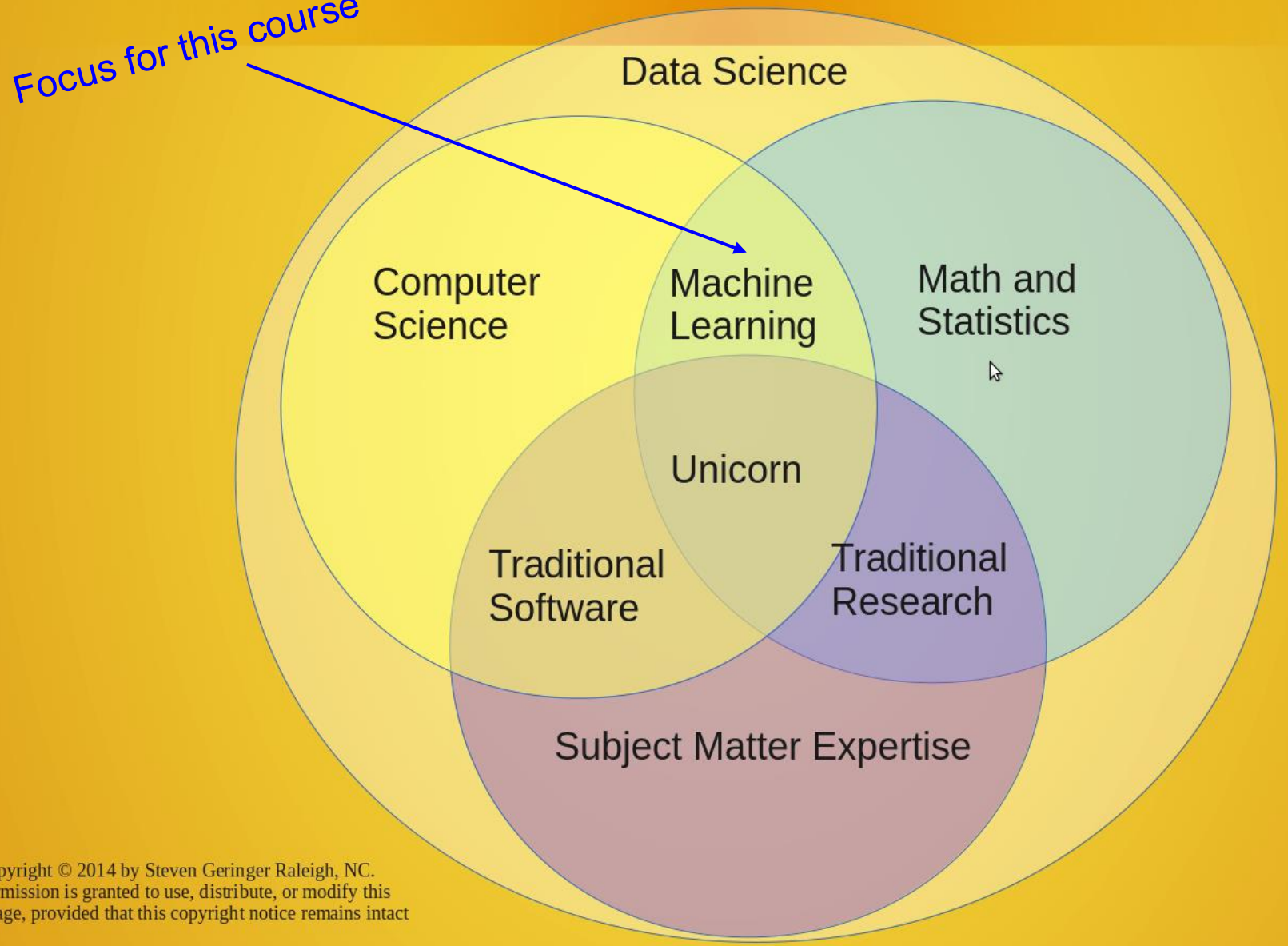
Extracting

- Emphasizes action on data
- Mining information

Math + Computer Science + Domain Knowledge



Data Science Venn Diagram v2.0

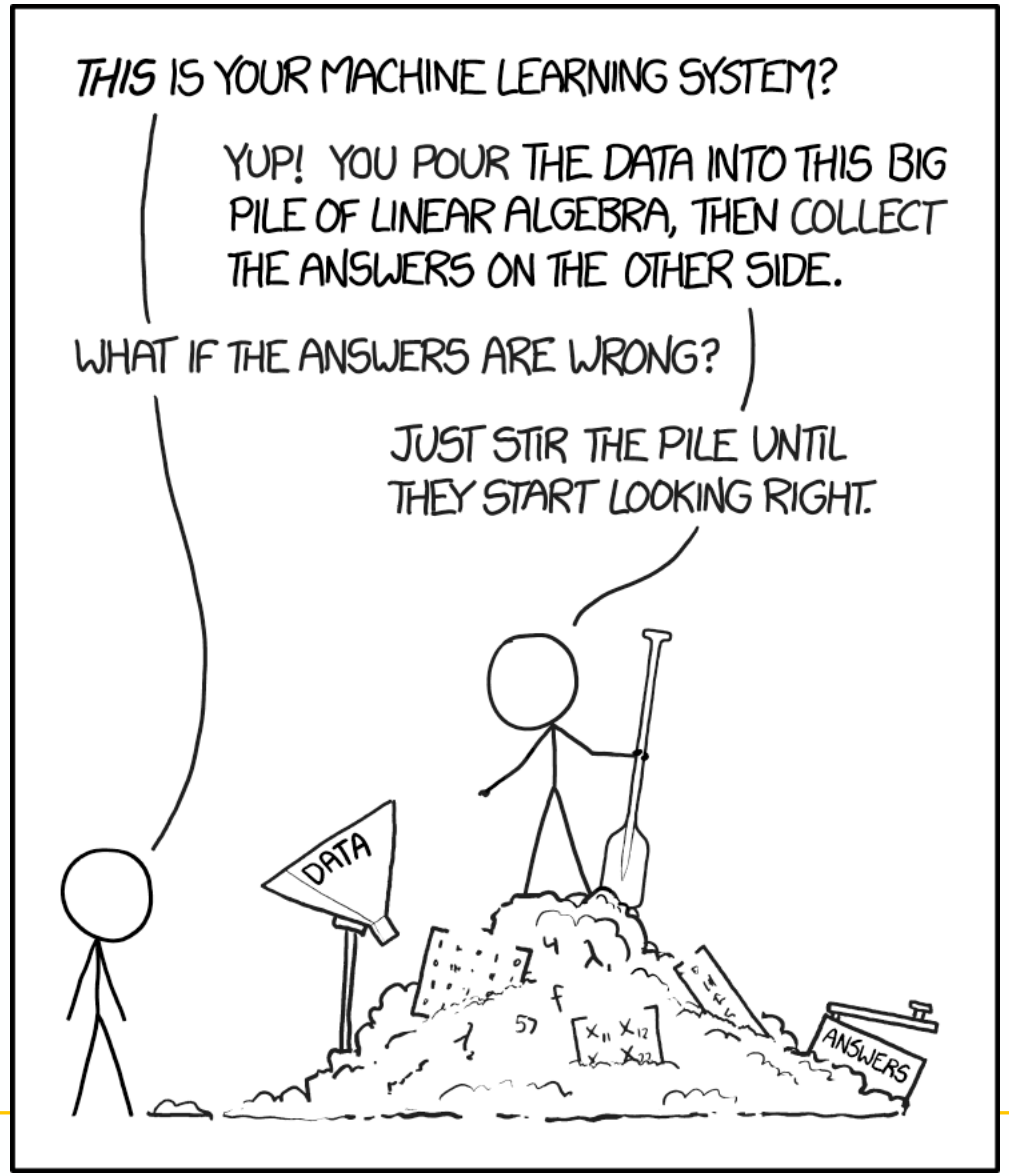


Copyright © 2014 by Steven Geringer Raleigh, NC.
Permission is granted to use, distribute, or modify this image, provided that this copyright notice remains intact

CS260 is not an entry level DS course

- "Math for Machine Learning"
- Meant to prepare students for 300-level data-oriented courses
  - Machine Learning (CS360)
  - NLP/Computational Linguistics (CS325)
  - Robotics (CS369)
  - Computer Vision
  - Computational Biology
  - Advanced ML courses at Penn
- Most of these are in Python so we will also cover advanced Python topics and libraries

There will be math!



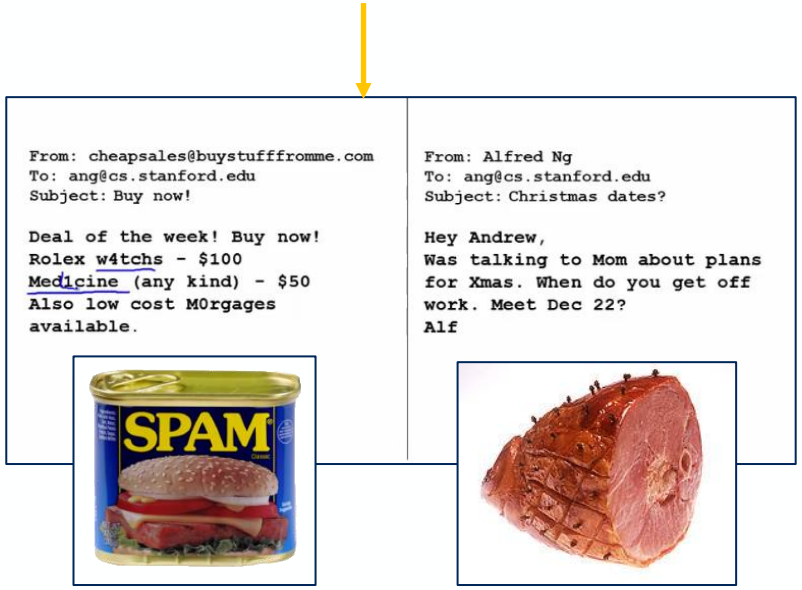
KEY COMPETENCIES FOR AN UNDERGRADUATE DATA SCIENCE STUDENT

- Computational and statistical thinking
- Mathematical foundations
- Model building and assessment
- Algorithms and software foundation
- Data curation
- Knowledge transference—communication and responsibility

From: De Veaux *et al*, 2017

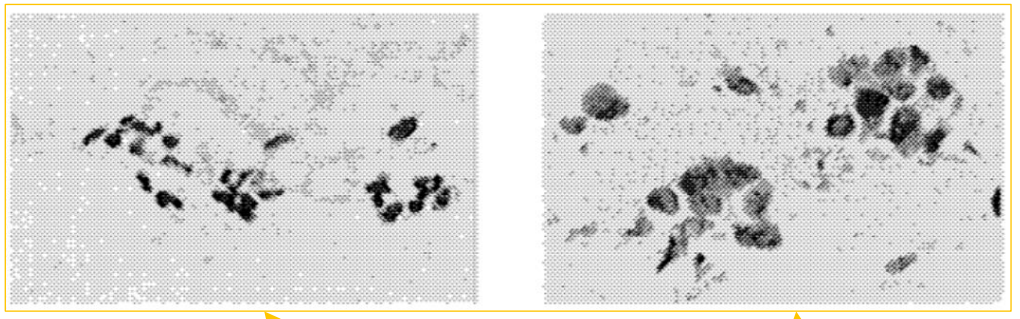
Learning from data: classification

- Email filtering (spam vs. not-spam)



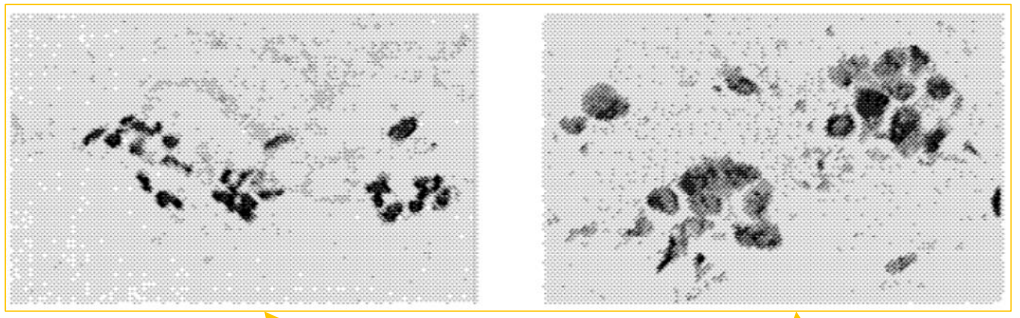
Images: Wikipedia, Matouš Havlena

Learning from data: classification

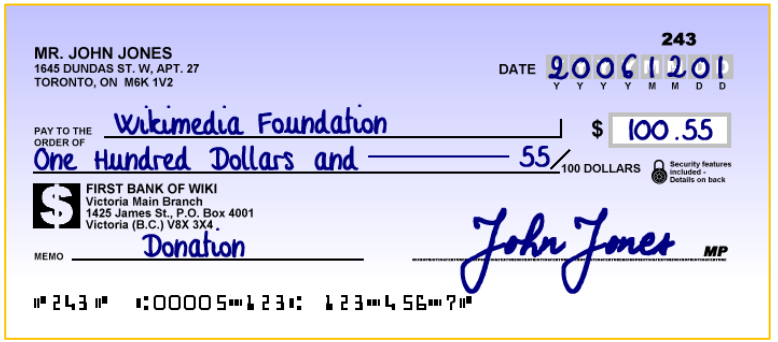


- Tumor detection (benign vs. malignant)

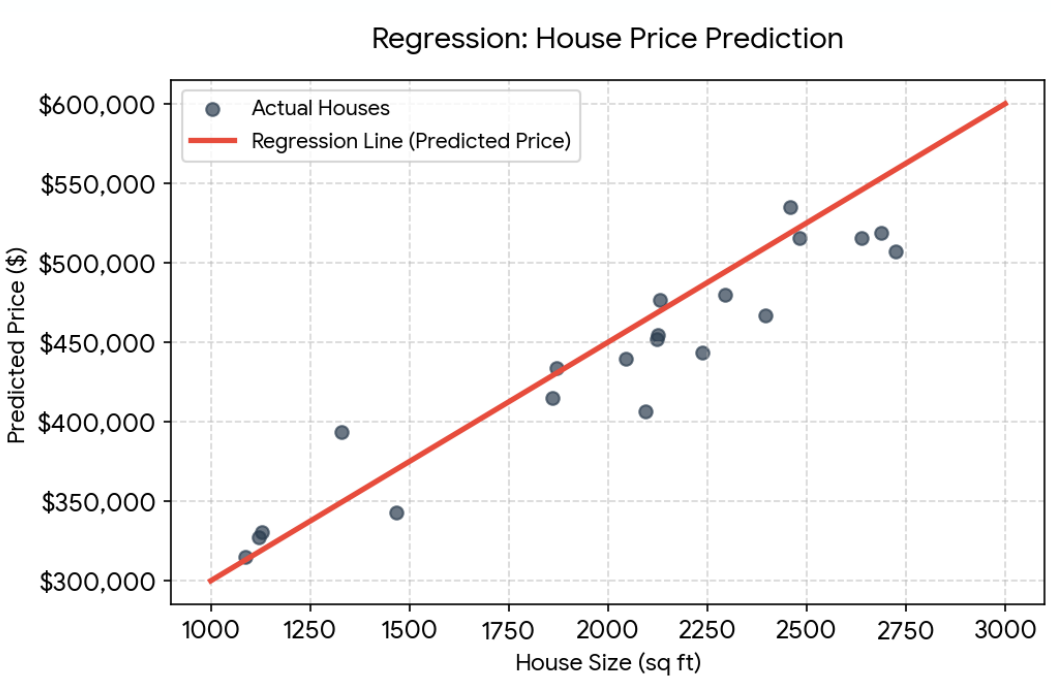
Learning from data: classification

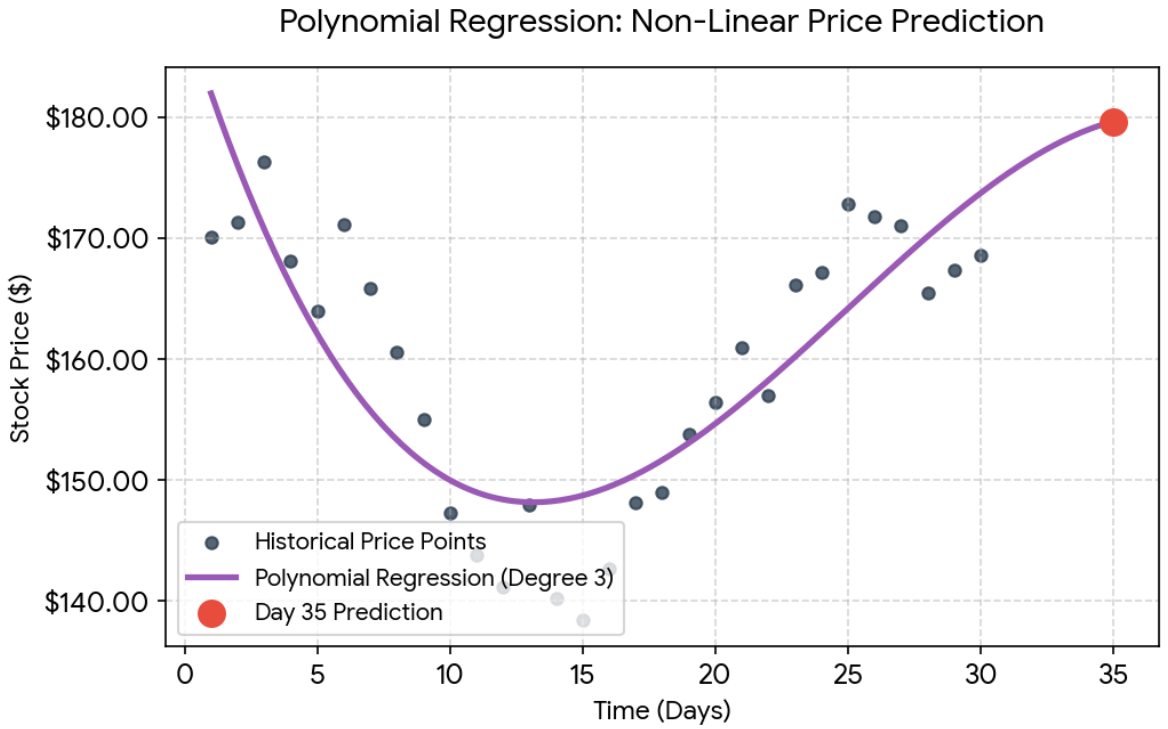


- Tumor detection (benign vs. malignant)
- Handwriting recognition (digits in a check)



Learning from data: regression





Statistical Tests

Do smokers weigh the same as non-smokers?

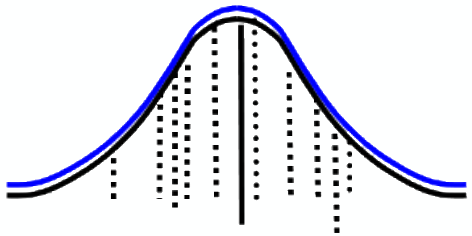
Wayne W. LaMorte, MD, PhD, MPH
Boston University School of Public Health

Statistical Tests

Do smokers weigh the same as non-smokers?

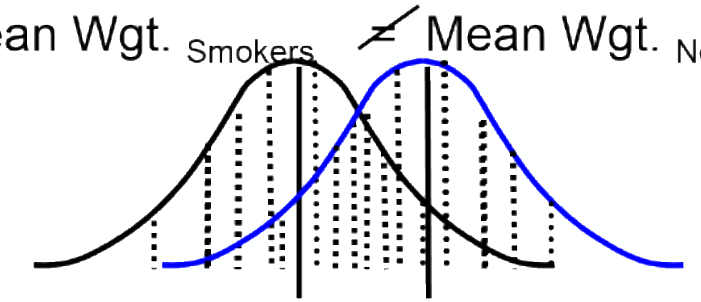
Null Hypothesis (H_0): the average weight does *not* differ

H_0: Mean Wgt. $_{smokers}$ = Mean Wgt. $_{Non\text{-}smokers}$



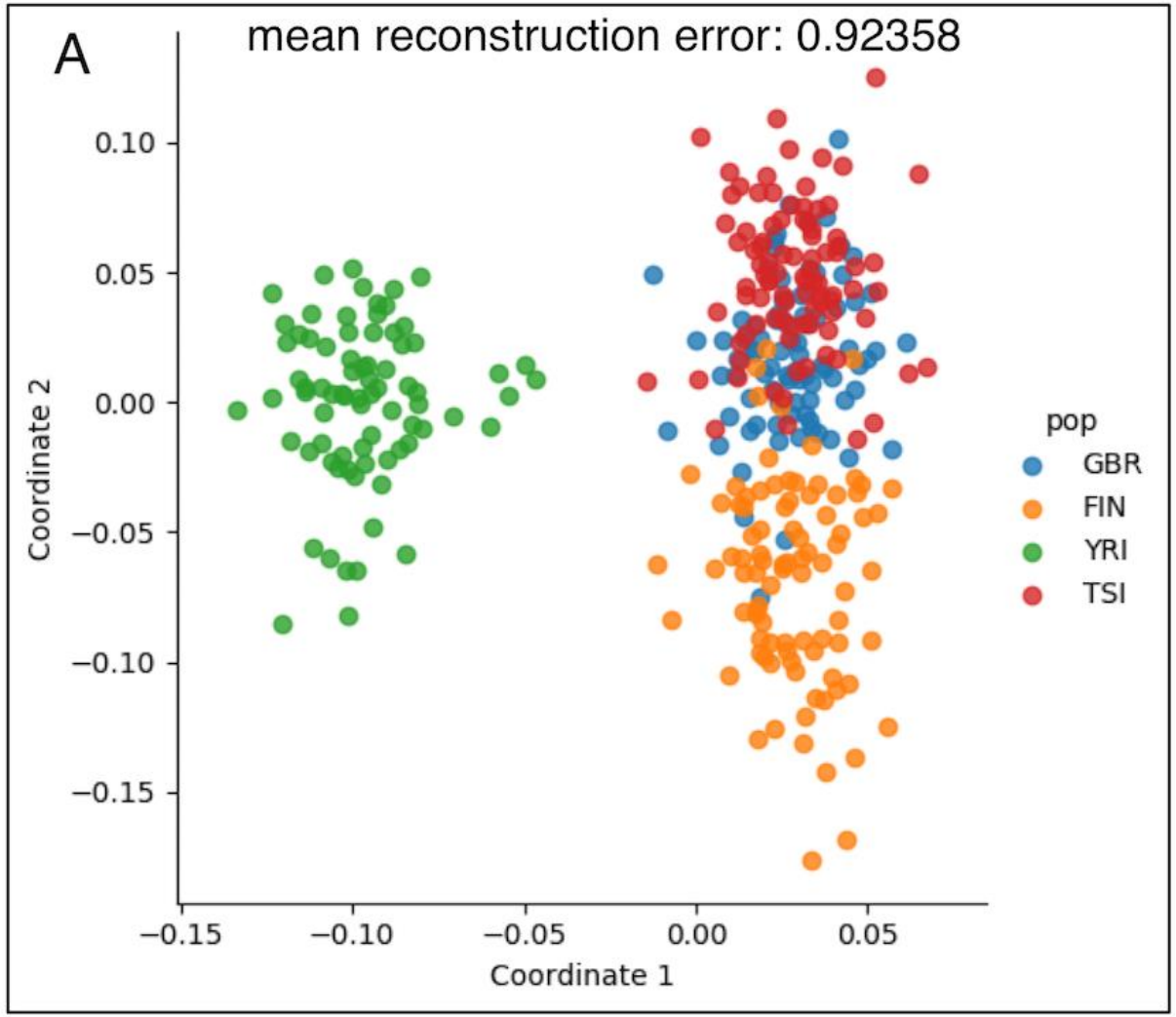
Alternative Hypothesis (H_A): the average weights differ

H_A: Mean Wgt. $_{Smokers}$ $\neq$ Mean Wgt. $_{Non\text{-}smokers}$



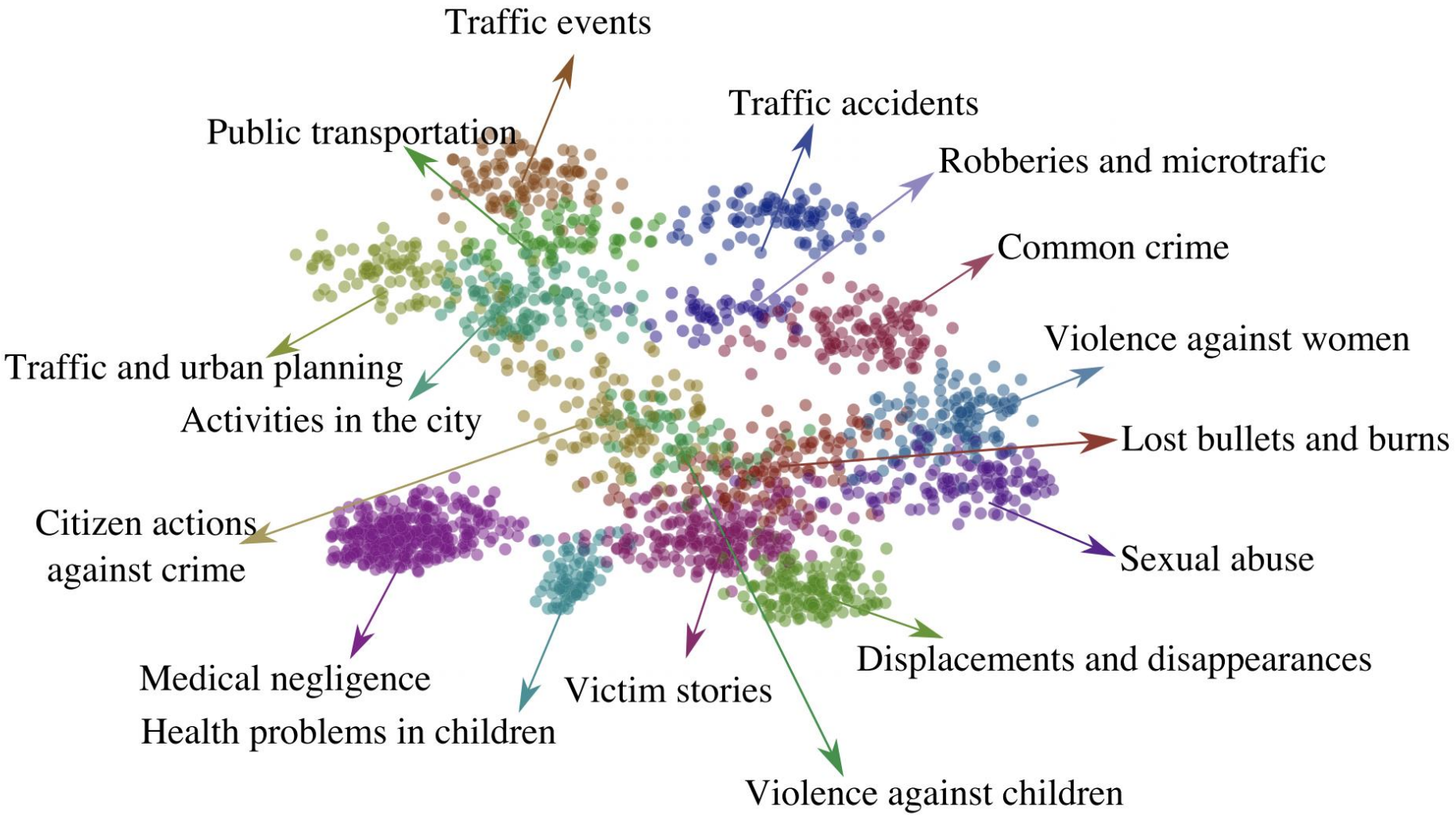
Wayne W. LaMorte, MD, PhD, MPH
Boston University School of Public Health

Data Visualization



"Expression reflects population structure", Brown *et al* (2018)

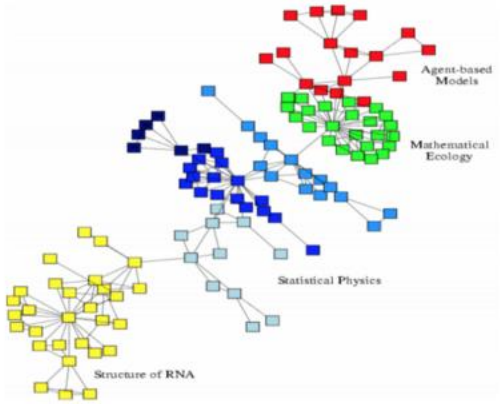
Clustering Data



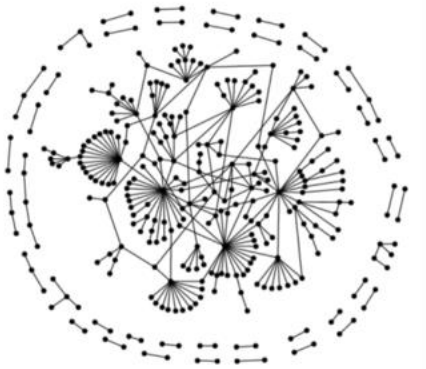
"Event detection in Colombian security Twitter news using fine-grained latent topic analysis", Vargas-Calderon *et al* (2019)

Learning from data: networks

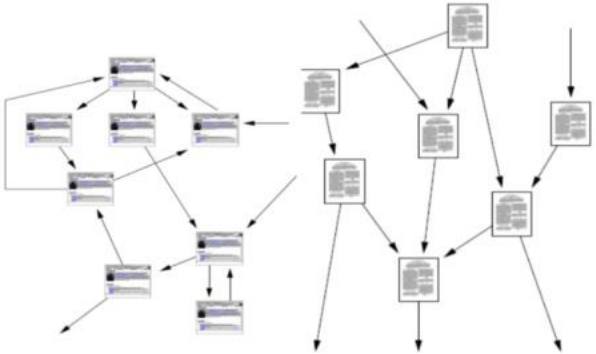
Social networks



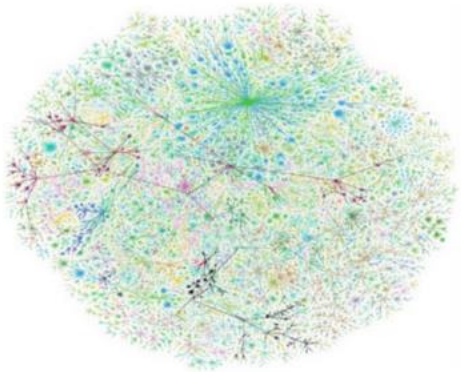
Economic networks



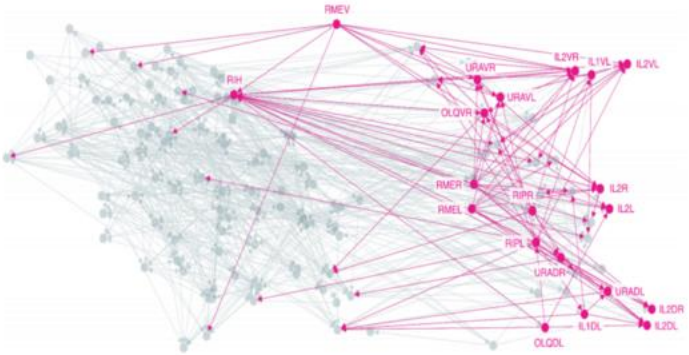
Biomedical networks



Information networks:
Web & citations



Internet



Networks of neurons

Natural Language Examples

- Text Classification
- Language Modeling
- Speech Recognition
- Caption Generation
- Machine Translation
- Document Summarization
- Question Answering
- Text Generation

Machine reading comprehension

Q: What was the theme?
A: "one world, one dream".

Q: What was the length of the race?
A: 137,000 km

Q: Was it larger than previous ones?
A: No

Q: Where did the race begin?
A: Olympia, Greece

Q: Is there anything notable about that place?
A: birthplace of Olympic Games

Q: Where did they go after?
A: Athens

Q: How many days was the race?
A: seven

Q: Did they visit any notable landmarks?
A: Panathinaiko Stadium

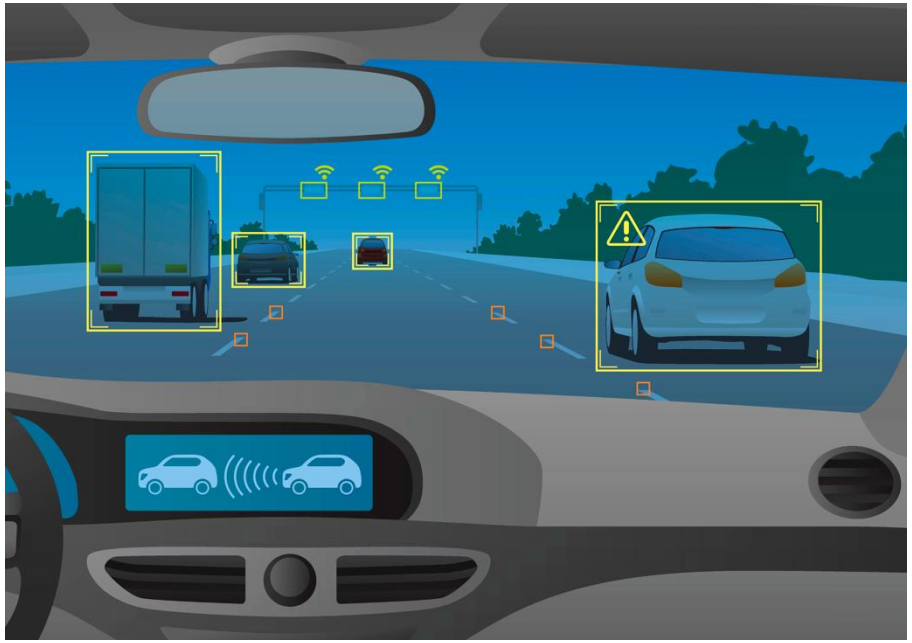
Q: And did they climb any mountains?
A:

Target answers: *unknown* or *yes*
Model answer: Everest

https://openai.com/blog/better-language-models/

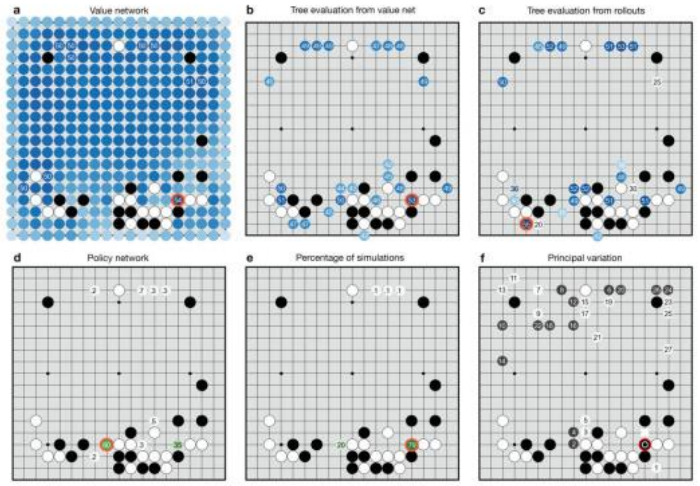
https://machinelearningmastery.com/applications-of-deep-learning-for-natural-language-processing/

Modern Machine Learning examples



Self-driving cars are in our present and futur

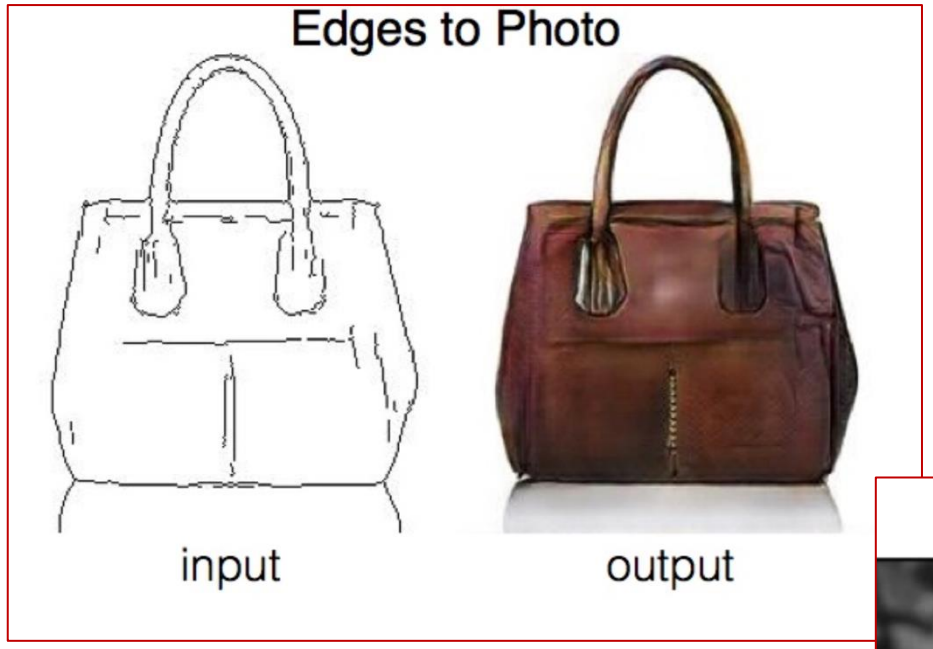
AlphaGo: moves humans never thought of



AlphaGo

Images: Scientific American

Modern Machine Learning examples



- Algorithms that learn how to create

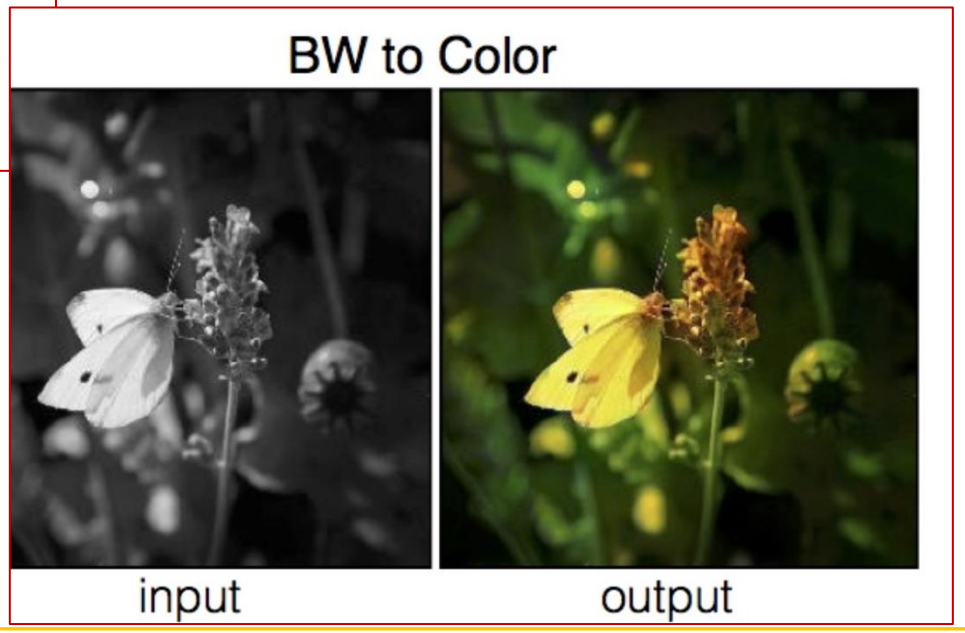


Image-to-Image Translation with Conditional Adversarial Nets (Nov 2016)

Generative Models

Generative models have also been used to create synthetic genomic data to maintain privacy

Generative Models

Style transfer with GANs

Stable Diffusion

"Surrealist painting of a floating island with giant clock gears, populated with mythical creatures."

Logistics

brynmawr.edu

Communication

Course webpage

https://cs.brynmawr.edu/cs280

Piazza:

Course communication
Ask & Answer Questions
Can post anonymously

Course Meetings: 10:10 – 11:30 T/Th

Live class
- Daily quizzes
- Primarily lectures
- Discussions & exercises
- Q/A
- Recorded

Pre-class readings:
- Expected to read assigned readings before class
- Distributed on course schedule

Lab
- 2:40-4 T
- Similar to lecture, but more emphasis on handouts


Homeworks
LEARNING BY DOING

Homeworks

Weekly through out semester

Submit individually

Autograder

Typically due Tuesday night

Late day policy

Submit after deadline, grade for assignment is capped at 75%
Can resubmit until the next midterm
Ill grade on: Wednesday, before Midterms, and Final

HW01 – available, due Tuesday 09/08

AI Policy

None

Grades in this class are largely based on in-person quizzes and exams, thus improper collaboration or AI use is not of serious conern. However, my goal is not to catch you at exam times, but to encourage you to establish good learning habits before that.

You are encouraged to discuss the lecture material, worksheets, handouts and problem sets with other students. Study groups are highly recommended and very productive by all previous student reports. However, the only “product” of your discussion should be your memory/understanding - you should write up solutions individually. You may not exchange written work or computer files.

You are permitted to use AI-tools or the internet in general, to clarify concepts, to get guidance on assignments, as long as you do so in a responsible manner. The bottom line is you should not use ANY tool to replace your own thinking or analysis. Beware however, that it is easy to underestimate the effort it takes. Understanding a solution given is very far away from being able to come up with one yourself. This is the real danger in using AI-tools as “learning tools”. Beware of the shortcuts you are taking.

Code should not be copied without permission from the author. If permission is given, code should be cited at the location it is used with a comment. If your solution is inspired by any outside resources (I understand that sometimes it is hard to not see things), you MUST cite.

(adapted from Dianna Xu)

Rubric

Participation	10%
Homework	10%
Midterms	25% (10 + 15)
Final	55%

2 Midterms – dates on website

Participation

During class:

- Discussion
- Asking questions

Asynchronous

- Active on Pizza

Assignment Logistics

Distribution

Piazza

Gradescope

Course Staff

Adam Poliak

apoliak@brynmawr.edu

5th year at BMC

Taught 5 DS/ML courses

Research:

- Natural Language Processing
- Data Science applied to text data

Worked as data scientist in industry

My job is to help you succeed!

Python demo

Python demo

Input
Random
Plotting
Numpy
dictionaries

Handout 1 – Part 1

Python style

Decompose code into natural functions

Avoid global variables (sometimes useful)

Include a file header with purpose, author, and date

Include headers for each function

No lines over 100 chars

Variable names implicitly show type

Include line breaks and comments!

Python style

"Snake-case" not "camel-case"

~~linearSearch~~

linear_search

Alphabetize imports and don't use "*"

~~from numpy import *~~

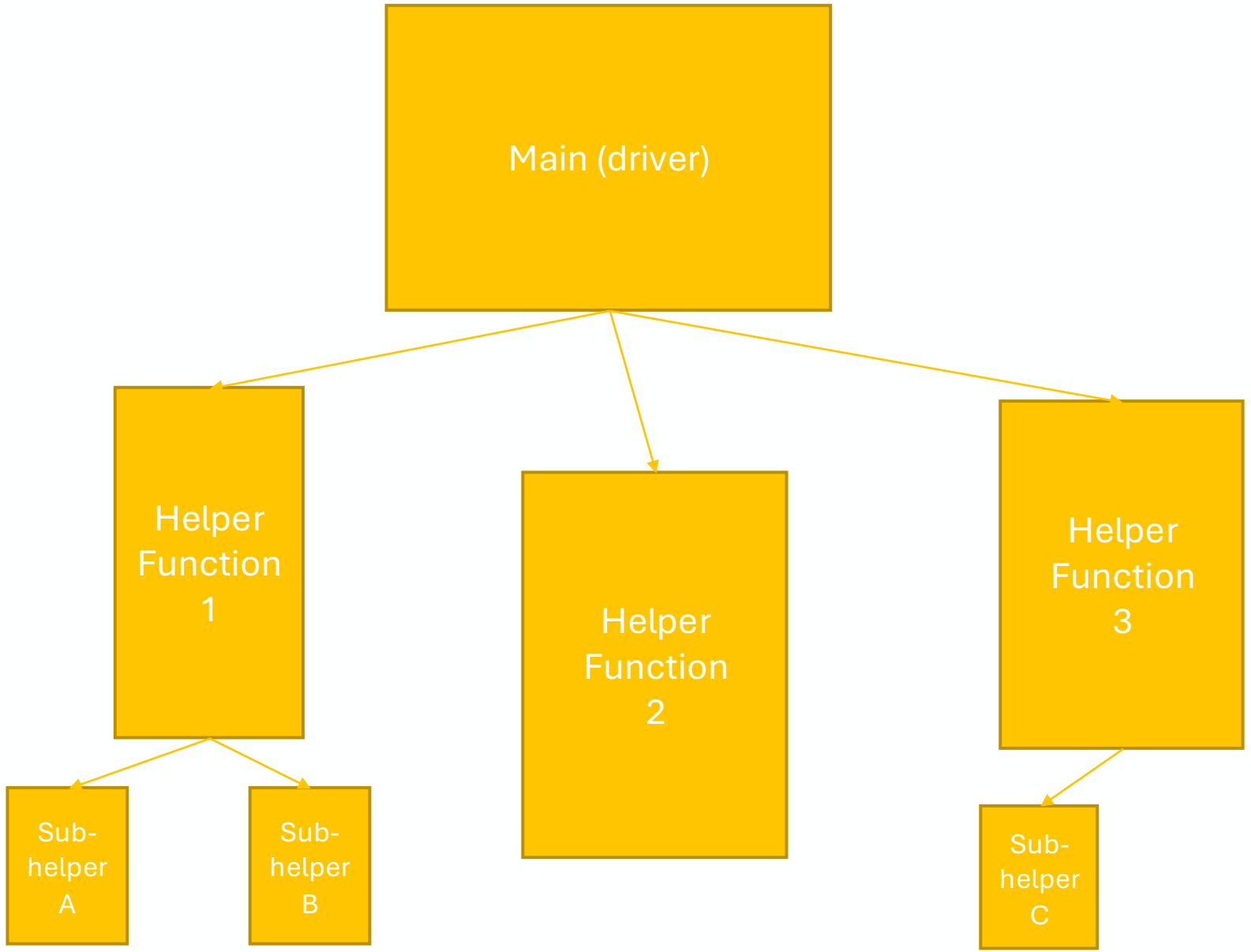
import numpy as np

Python style examples

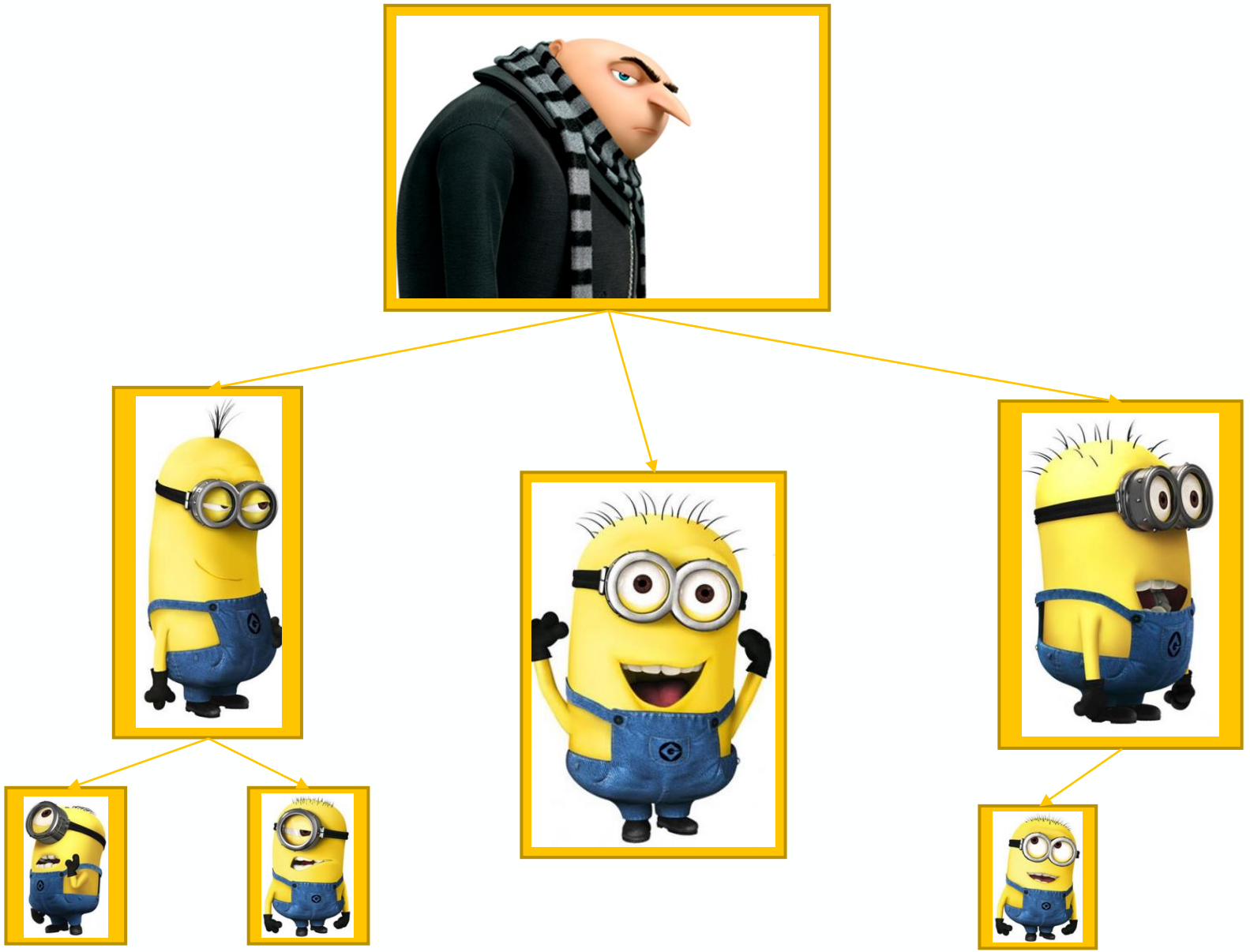
```
"""
Ask the user for their name and welcome them to CS21.
Author: Sara Mathieson
Date: 9/7/18
"""

def main():

    # ask user for their name and print greeting
    name = input("Enter your name: ")
    print("Hello", name, "!")

main()
```

```
def factorial(n):
    """
    Given a non-negative integer n, return n! = n*(n-1)*(n-2)....3*2*1.
    """
    fact = 1 # set up an accumulator variable
    for i in range(n):
        fact = fact * (i+1) # accumulator pattern
    return fact
```

Structure of main and “helper” functions



Structure of main and “helper” functions



Reminder: steps of top-down-design (TDD)

Reminder: steps of top-down-design (TDD)

1) Design a **high-level main function** that captures the basic idea of the program.

Reminder: steps of top-down-design (TDD)

1) Design a **high-level main function** that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.

Reminder: steps of top-down-design (TDD)

1) Design a **high-level main function** that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.

3) **"Stub" out the functions**. This means that they should work and return the correct type so that your code runs, but they don't do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

Reminder: steps of top-down-design (TDD)

1) Design a **high-level main function** that captures the basic idea of the program.

2) As you're writing/designing main, think about which details can be **abstracted into small tasks**. Make names for these functions and write their signatures below main.

3) **"Stub" out the functions**. This means that they should work and return the correct type so that your code runs, but they don't do the correct task yet. For example, if a function should return a list, you can return []. Or if it returns a boolean, you can return False.

4) Iterate on your design until you have a working main and stubbed out functions. Then start **implementing** the functions, starting from the "bottom up".

Reasons to use TDD

Creates code that is easier to implement, debug, modify, and extend

Avoids going off in the wrong direction (i.e., implementing functions that are not useful or don't serve the program)

Creates code that is easier for you or someone else to read and understand later on

Don’t be like them!

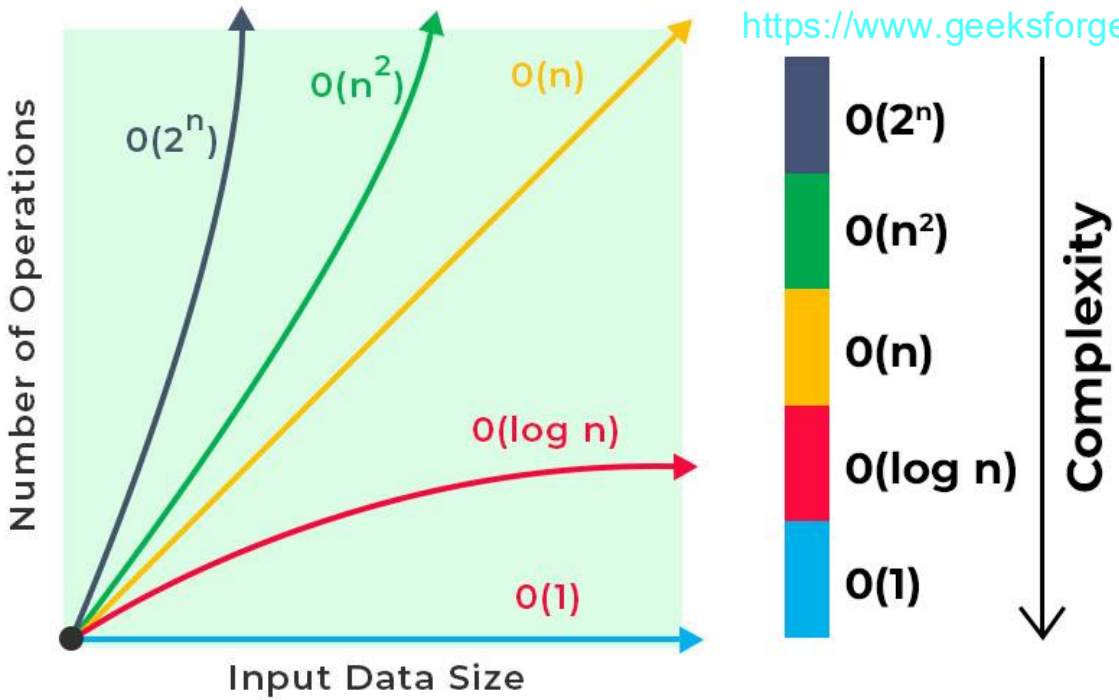
Algorithm Runtime/Big-O notation Review

- Largely dependent on the number of data points and data structures in use
- Constant time operations (**O(1)**): accessing values in an array, dictionary, etc.
- Iterating through n items: **O(n)**
- Sequential steps: **add** their runtime
- Nested operations: **multiply** their runtime
- Big-O notation: ignore scalars and small terms, assume worst case

Big-O notation Practice

What is the big-O runtime of:

- Dividing a number n by 10 until we get 1
- Guessing an n-digit password



Object Oriented Programming

Classes allow us to encapsulate common data structures and actions so we don't have to define them over and over again

Classes in Python represent the same idea as classes in Java

Example: say we have two classes: **Point** and **Circle**

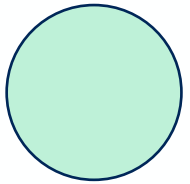
Object Oriented Programming

Example: say we have two classes: **Point** and **Circle**

We can create a new *instance* of a class using the:

constructor

dot = Circle(Point(x,y), r)



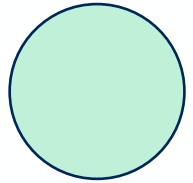
Object Oriented Programming

Example: say we have two classes: **Point** and **Circle**

We can create a new *instance* of a class using the:

constructor

```
dot = Circle(Point(x,y), r)
```



We can access the instance's data using *methods*

```
r = dot.get_radius()
```

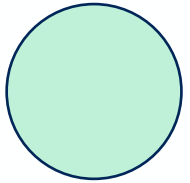
Object Oriented Programming

Example: say we have two classes: **Point** and **Circle**

We can create a new *instance* of a class using the:

constructor

```
dot = Circle(Point(x,y), r)
```

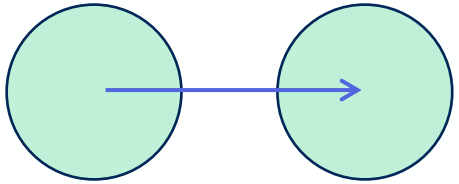


We can access the instance's data using *methods*

```
r = dot.get_radius()
```

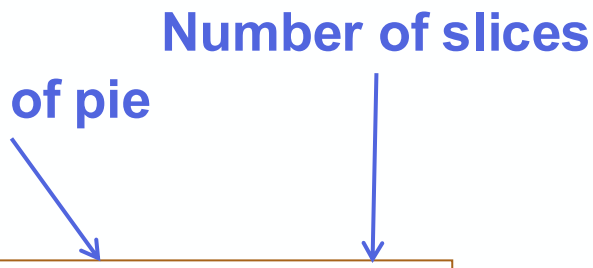
We can use/modify class instances using *methods*

```
dot.move(dx,dy)
```



Motivation for classes: LOLs

List-of-lists let us keep track of things that should be “together”, but they get cumbersome to modify:




```
>>> pie_lst = [["apple",8], ["cherry",8], ["chocolate",8]]
>>>
>>> pie_lst[2][1] -= 1
>>>
>>> pie_lst
[['apple', 8], ['cherry', 8], ['chocolate', 7]]
```

Motivation for classes: encapsulation and abstraction

Neither encapsulated (data for one student is spread over multiple objects), nor abstract

```
name_lst = ["Kendre", "Rohan", "Ayaka", "Maleyah"]
year_lst = [2020, 2021, 2020, 2021]
name = name_lst[0]
year = year_lst[0]
```

Motivation for classes: encapsulation and abstraction

Neither encapsulated (data for one student is spread over multiple objects), nor abstract

```
name_lst = ["Kendre", "Rohan", "Ayaka", "Maleyah"]
year_lst = [2020, 2021, 2020, 2021]
name = name_lst[0]
year = year_lst[0]
```

Encapsulated (student is represented as one thing, a list), but not abstract

```
kendre = ["Kendre", 2020, ["cs35","act1","relg43","span1"]]
name = kendre[0]
year = kendre[1]
```

Motivation for classes: encapsulation and abstraction

Neither encapsulated (data for one student is spread over multiple objects), nor abstract

```
name_lst = ["Kendre", "Rohan", "Ayaka", "Maleyah"]
year_lst = [2020, 2021, 2020, 2021]
name = name_lst[0]
year = year_lst[0]
```

Encapsulated (student is represented as one thing, a list), but not abstract

```
kendre = ["Kendre", 2020, ["cs35","act1","relg43","span1"]]
name = kendre[0]
year = kendre[1]
```

Both abstract and encapsulated

Should be:
get_name()
get_year()

```
kendre = Student("Kendre", 2020)
name = kendre.getName()
year = kendre.getYear()
```

Advantages of encapsulation/abstraction

Interface (how you interact with something) is consistent even if the internal details change.

1) If you change the engine in your car, you still drive it the same way – don't need to know how the engine works.

2) In online shopping you have a "Cart", which is an abstract concept and is roughly the same across sites. Probably represented as a list underneath but user doesn't need to know.

“Pie” class example

```
class Pie: # class names should be capitalized

    # must use init for the constructor
    def __init__(self, flavor):
        """Constructor for the Pie class."""
        # in the constructor, define the data (i.e. self.data)
        # data are called: attributes or instance variables
        self.flavor = flavor
        self.slices = 8

    def get_slices(self):
        """Return the number of slices left (int)."""
        return self.slices

    def get_flavor(self):
        """Return the flavor of the pie (str)."""
        return self.flavor
```

“Pie” class example

```
def serve(self):
    """If there is at least one slice left, reduce the number of slices."""
    if self.slices > 0:
        print("Here is a slice of %s pie!" % self.flavor)
        self.slices -= 1
    else:
        print("Sorry, there is no more %s pie!" % self.flavor)

def __str__(self):
    """Return a string representation of a pie."""
    s = "%s pie has %i slices left!" % (self.flavor, self.slices)
    return s
```

better to use: f“{self.flavor} pie has {self.slices} left!”

“Pie” class example

```
def main():
    pie1 = Pie("apple")
    print(pie1) # __str__ is automatically called when we call print(..)
    for i in range(12):
        pie1.serve()
    print(pie1.get_slices())
    print(pie1.get_flavor())
    print(pie1)

    pie2 = Pie("pumpkin")
    print(pie2)
    pie2.serve()
    print(pie2)
```

```
apple pie has 8 slices left!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Here is a slice of apple pie!
Sorry, there is no more apple pie!
Sorry, there is no more apple pie!
Sorry, there is no more apple pie!
Sorry, there is no more apple pie!
0
apple
apple pie has 0 slices left!
pumpkin pie has 8 slices left!
Here is a slice of pumpkin pie!
pumpkin pie has 7 slices left!
```

TwitterUser class

```
class TwitterUser: # only time camel case is okay!

    # constructor
    def __init__(self, name, curr_following, curr_followers):
        self.name = name
        self.following = curr_following
        self.followers = curr_followers

    def add_follower(self): # always have to use self!
        self.followers += 1
        # TODO we could make this better by creating a list of followers who
        # are themselves instances of TwitterUser

    def follow(self):
        self.following += 1

    def __str__(self):
        # must return a string, not print a string!
        return "name: %s\nnum following: %i\nnum followers: %i" % (self.name, \
            self.following, self.followers)
```

Handout 1 – part two

Find and work with a partner

Recap Die class

Defining the Constructor: builds an instance of the class (self), and initializes all instance variables (self.xxx)

```
class Die:

    def __init__(self, num_sides):
        """Construct a new die with the given number of sides."""
        self.sides = num_sides
        self.value = 1 # default starting value
```

Recap Die class

Defining the Constructor: builds an instance of the class (self), and initializes all instance variables (self.xxx)

```
class Die:

    def __init__(self, num_sides):
        """Construct a new die with the given number of sides."""
        self.sides = num_sides
        self.value = 1 # default starting value
```

Using the Constructor: assign the new object to a variable, making the “self” placeholder a concrete instance

```
def main():
    # create 8-sided dice
    die1 = Die(8)
    die2 = Die(8)
```

Recap Die class

Defining Methods: always use “self” as the first argument (placeholder for the instance). Getters are a type of method that return instance variables or their derivatives.

```
def getValue(self):
    """Getter for the die's current value."""
    return self.value

def roll(self):
    """Choose a new random value for the die, i.e. roll it."""
    self.value = random.randrange(1,self.sides+1)
```

Recap Die class

Defining Methods: always use “self” as the first argument (placeholder for the instance). Getters are a type of method that return instance variables or their derivatives.

```
def getValue(self):
    """Getter for the die's current value."""
    return self.value

def roll(self):
    """Choose a new random value for the die, i.e. roll it."""
    self.value = random.randrange(1,self.sides+1)
```

Using Methods: instance.method(…), don’t use self

```
# roll both until we get the same value
same = False
while not same:
    die1.roll()
    die2.roll()
    print(die1)
    print(die2)
    print()
    # check if the values are the same
    same = (die1.getValue() == die2.getValue())
```

Recap Die class

Defining the __str__ method: no print(..) statements!
Build and return a single string. (no arguments besides self)

```
def __str__(self):
    """String representation of the die (with current value)."""
    return "%d-sided die, current value: %d" % (self.sides, self.value)
```

Recap Die class

Defining the __str__ method: no print(..) statements!
Build and return a single string. (no arguments besides self)

```
def __str__(self):
    """String representation of the die (with current value)."""
    return "%d-sided die, current value: %d" % (self.sides, self.value)
```

Using the __str__ method: simply call print(instance)!

```
print(die1)
print(die2)
```

```python
# open(..) returns a file object (called an TextIOWrapper but think: file)
c_file = open("colleges.txt", 'r') # 'r' for read, 'w' for write

enroll_lst = []

# one way to read a file: loop through each line of the file
for line in c_file:
    # split breaks up the line on spaces, it is a method that returns a list
    tokens = line.split()

    # extract information from specific tokens
    name = tokens[0]
    enroll = int(tokens[1])
    enroll_lst.append(enroll)

# always remember to close your files!
c_file.close()
```

colleges.txt

```
Amherst 1792
Bates 1792
Bowdoin 1806
BrynMawr 1709
Colby 1815
Davidson 1950
HarveyMudd 735
Haverford 1290
Middlebury 2526
Pomona 1663
Reed 1411
Smith 2600
Swarthmore 1620
Vassar 2450
Wellesley 2474
Williams 2099
```

Example of reading in data

File reading demo

```
import csv
import numpy as np

# 1) read line by line
fb_file = open("data/facebook_users.csv", 'r') # 'r' for read mode
for line in fb_file:
    tokens = line.split(",") # split on comma
    year = int(tokens[0])
    num_users = int(tokens[1])
    print(year, num_users)
fb_file.close()
```

File reading demo

```
import csv
import numpy as np

# 1) read line by line
fb_file = open("data/facebook_users.csv", 'r') # 'r' for read mode
for line in fb_file:
    tokens = line.split(",") # split on comma
    year = int(tokens[0])
    num_users = int(tokens[1])
    print(year, num_users)
fb_file.close()

# 2) csv reader
with open("data/facebook_users.csv", 'r') as fb_file:
    csv_reader = csv.reader(fb_file)
    for line in csv_reader:
        print(line)
```

File reading demo

```
import csv
import numpy as np

# 1) read line by line
fb_file = open("data/facebook_users.csv", 'r') # 'r' for read mode
for line in fb_file:
    tokens = line.split(",") # split on comma
    year = int(tokens[0])
    num_users = int(tokens[1])
    print(year, num_users)
fb_file.close()

# 2) csv reader
with open("data/facebook_users.csv", 'r') as fb_file:
    csv_reader = csv.reader(fb_file)
    for line in csv_reader:
        print(line)

# 3) load into numpy array
data = np.loadtxt("data/facebook_users.csv", dtype=int, delimiter=",")
print(data)
```

Numpy

Numerical Python

Designed for fast computation on arrays

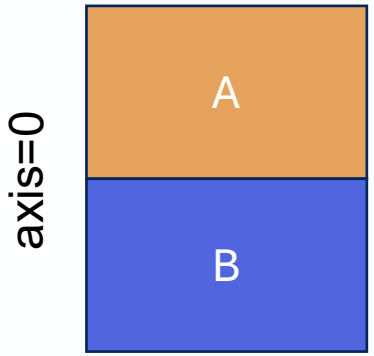
Implemented in C underneath

pip3 install numpy (on the terminal) OR

python3 –m pip install numpy

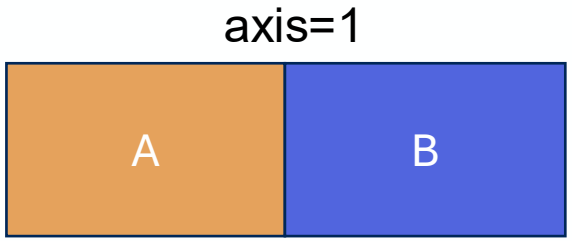
Numpy concatenation

np.concatenate((A,B), axis=0)



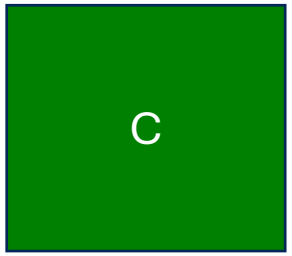
must match along axis 1

np.concatenate((A,B), axis=1)

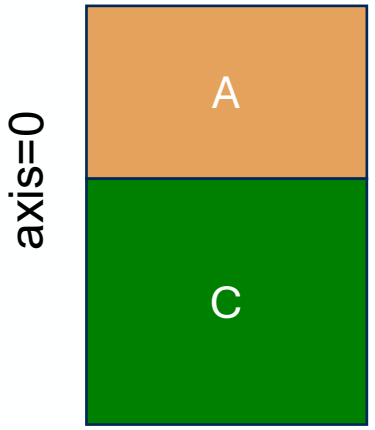


must match along axis 0

Numpy concatenation

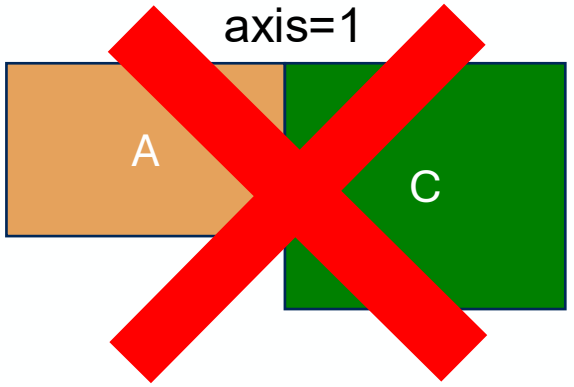


np.concatenate((A,C), axis=0)



must match along axis 1

np.concatenate((A,C), axis=1)



Error: must match along axis 0!

Numpy concatenation example

```
a = [[3,4,2],[7,8,9],[2,1,0]]
b = [[4,9,7],[3,0,1],[3,8,4]]

a_arr = np.array(a)
b_arr = np.array(b)
```

```
>>> a_arr
array([[3, 4, 2],
       [7, 8, 9],
       [2, 1, 0]])
>>> b_arr
array([[4, 9, 7],
       [3, 0, 1],
       [3, 8, 4]])
```

```
>>> many_rows = np.concatenate((a_arr,b_arr), axis=0)
>>>
>>> many_rows
array([[3, 4, 2],
       [7, 8, 9],
       [2, 1, 0],
       [4, 9, 7],
       [3, 0, 1],
       [3, 8, 4]])
```

```
>>> many_rows.shape
(6, 3)
>>> many_cols.shape
(3, 6)
```

```
>>> many_cols = np.concatenate((a_arr,b_arr), axis=1)
>>>
>>> many_cols
array([[3, 4, 2, 4, 9, 7],
       [7, 8, 9, 3, 0, 1],
       [2, 1, 0, 3, 8, 4]])
```

TODO

Reading: **MML Chap 1**

Read over **HW 1**

Sign up for Piazza & Gradescope