Algorithms and Data Structures for Data Science

Introduction

CS 277
Brad Solomon

January 17, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

Get to know each other through brief introductions

Discuss class logistics and expectations

Begin reviewing programming and Python fundamentals

Who am I?

Brad Solomon

Teaching Assistant Professor, Computer Science

2233 Siebel Center for Computer Science

Email: bradsol@illinois.edu

Office Hours:

Thursdays, 11:00 - 12:00 PM

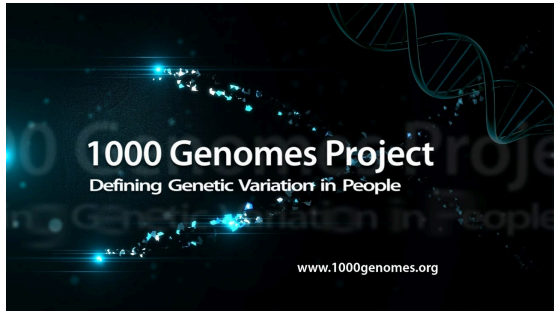
(Details are on the website)

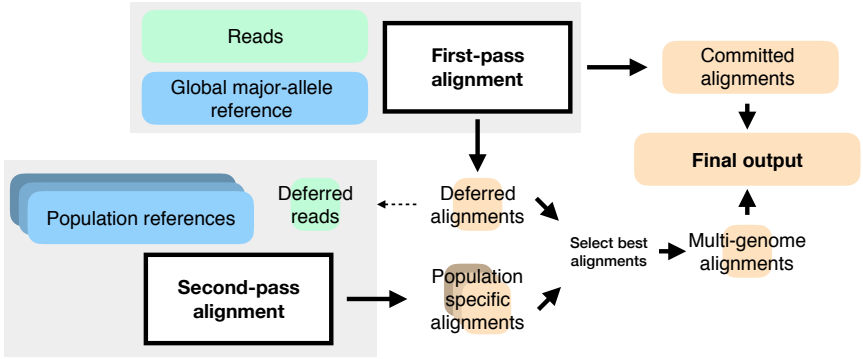
… can also make an appointment directly

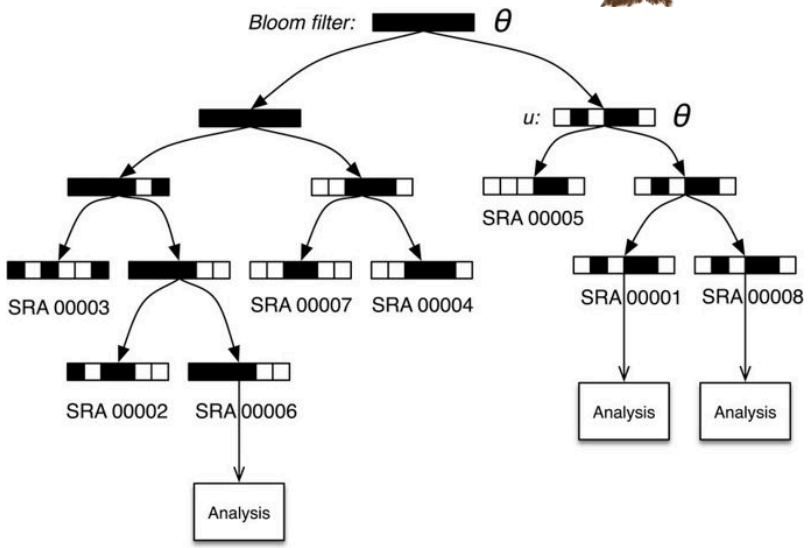
Who am I?

ENA
European Nucleotide Archive

GTEx







Fast search of thousands of short read sequencing experiments. Brad Solomon and Carl Kingsford. *Nature Biotech* 2016

Reducing reference bias using multiple population reference genomes. Chen et al. *Genome Biology* 2021

Course Staff Introductions

Who are you?

Feel free to introduce yourself on Piazza:

https://piazza.com/illinois/spring2024/cs277

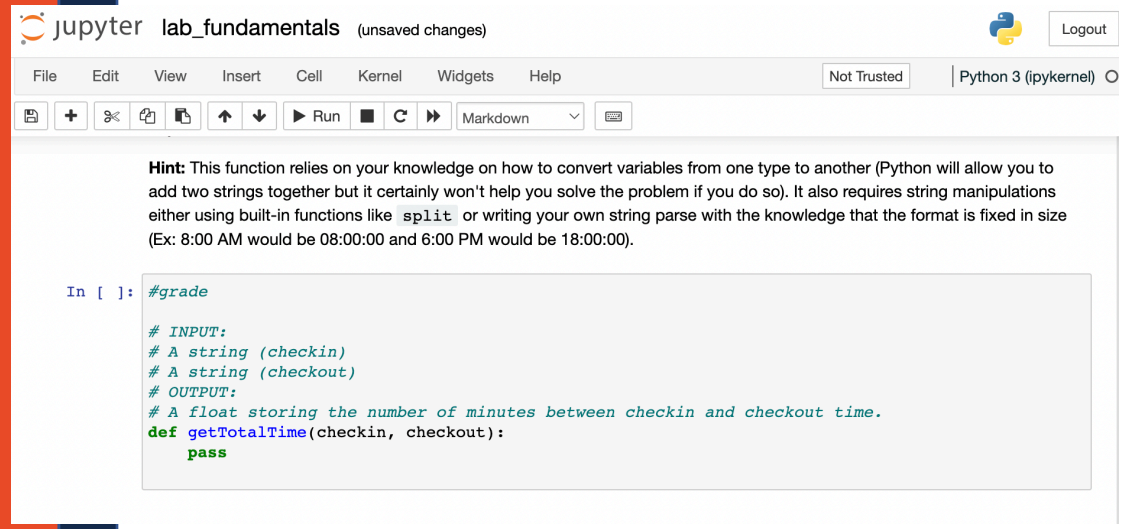
Freely talk with each other on Discord**

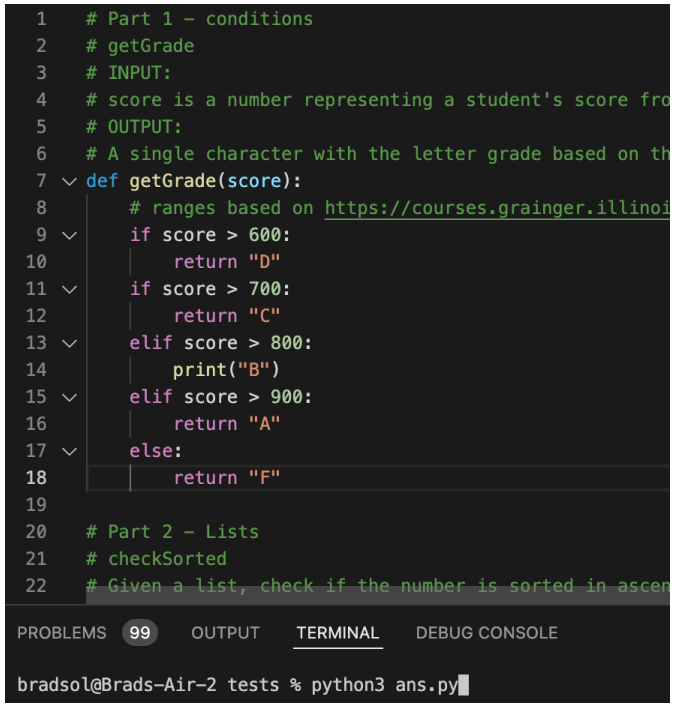
Introduce yourself when asking a question

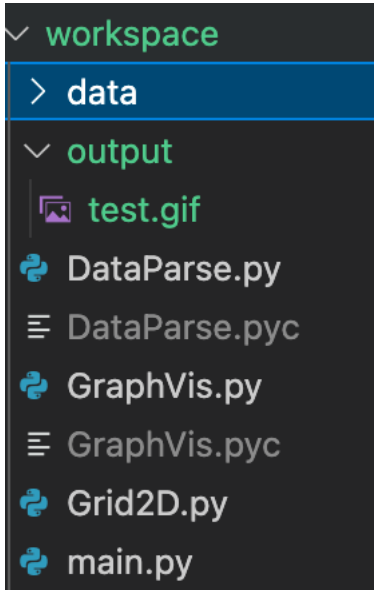
Stop by my office hours at some point this semester!

What will you get out of this class?

Navigate, organize, and run moderately complex Python projects







Why should you care?

Navigate, organize, and run moderately complex Python projects

```
TeamProject
├── UserInterface
│   ├── Mobile
│   ├── HTML
├── BackEnd
│   ├── BigData
│   ├── UserList
├── Algorithm
│   ├── SecretSort
│   ├── UserHash
```

```
#include <vector>

#include "util/coloredout.h"

#include "cs225/point.h"

using std::vector;
using std::string;
using std::ostream;
using std::cout;
using std::endl;
```

```
bradsol@Brads-Air-2 code % python3 debug.py
Use print statements, break statements, and return statements to debug errors!
We've given a few examples of print statements below.
Brad got 799 points and got a  D
Harsh got 800 points and got a  D

Now let's check ascending order
False

Traceback (most recent call last):
  File "debug.py", line 114, in <module>
    print(removeOdds(l1))
  File "debug.py", line 46, in removeOdds
    val = list_1d[i]
IndexError: list index out of range
```

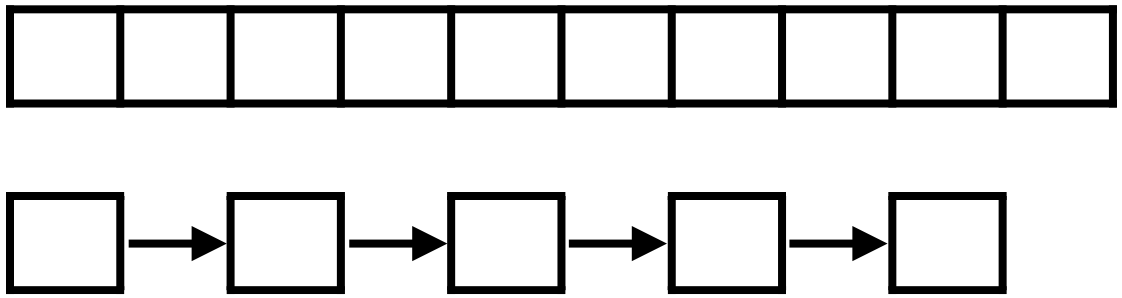
FASTQ-to-FASTA

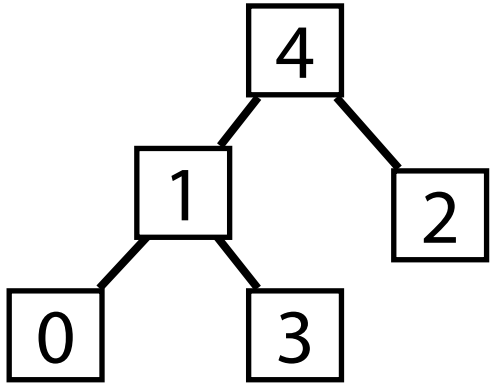
```
$ fastq_to_fasta -h
usage: fastq_to_fasta [-h] [-r] [-n] [-v] [-z] [-i INFILE] [-o OUTFILE]

version 0.0.6
   [-h]         = This helpful help screen.
   [-r]         = Rename sequence identifiers to numbers.
   [-n]         = keep sequences with unknown (N) nucleotides.
                  Default is to discard such sequences.
   [-v]         = Verbose - report number of sequences.
                  If [-o] is specified,  report will be printed to STDOUT.
                  If [-o] is not specified (and output goes to STDOUT),
                  report will be printed to STDERR.
   [-z]         = Compress output with GZIP.
   [-i INFILE]  = FASTA/Q input file. default is STDIN.
   [-o OUTFILE] = FASTA output file. default is STDOUT.
```

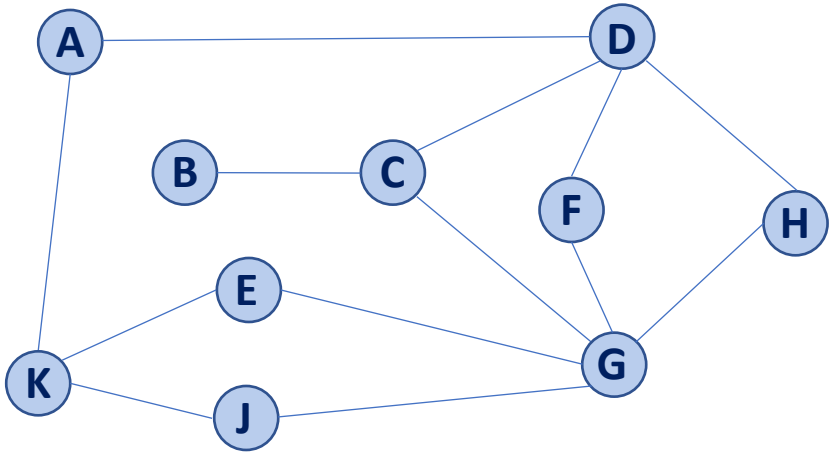
Taken from FastX-Toolkit (hannonlab.cshl.edu)

What will you get out of this class?

Understand foundational data structures and algorithms

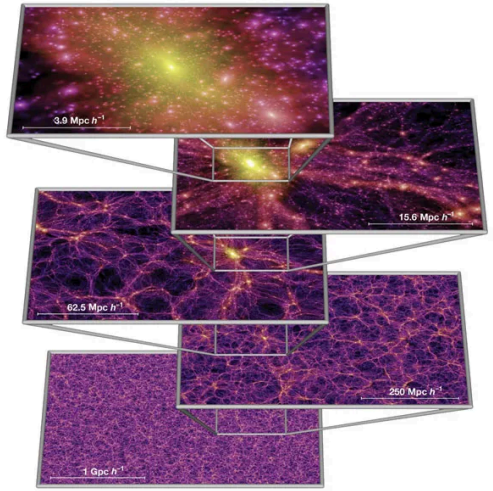




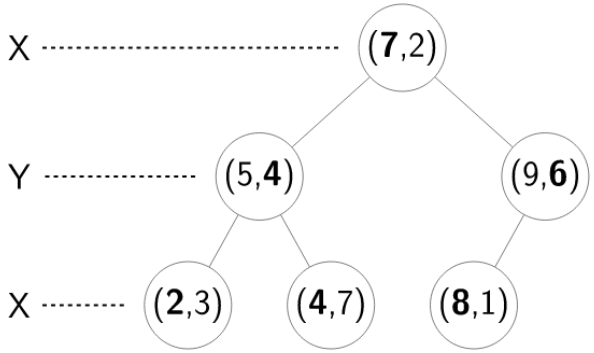


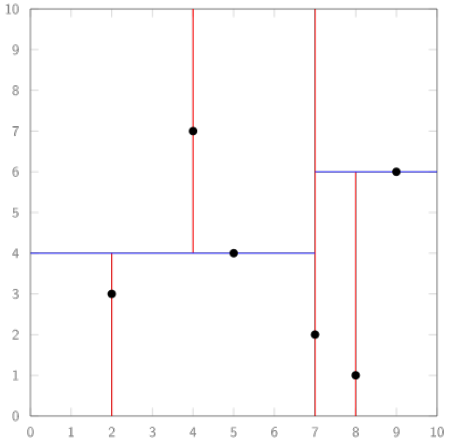
Why should you care?

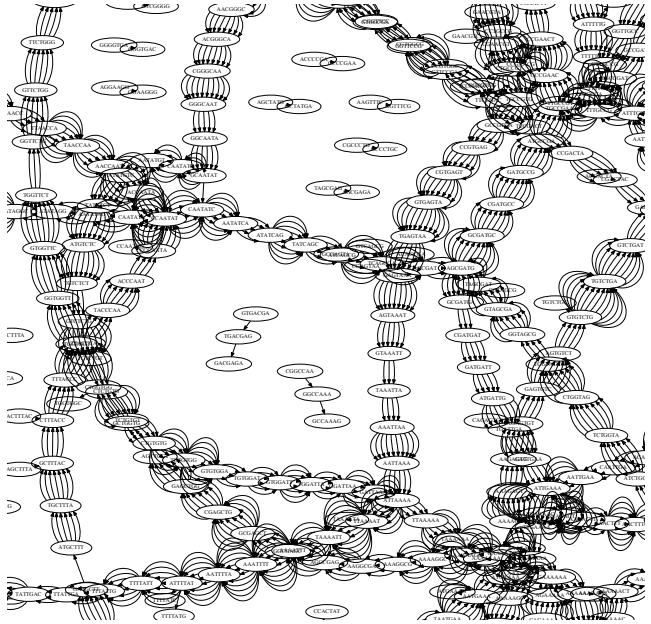
Understand foundational data structures and algorithms




```
Query: 161 atatcaccacgtcaaaggtgactccaactcca---ccactccattttgttcagataatgc 217
                ||||||||||||||||||||||||||||||  |     | |   || |||||||||||||
Sbjct: 481 atatcaccacgtcaaaggtgactccaact-tattgatagtgttttatgttcagataatgc 539
```







What will you get out of this class?

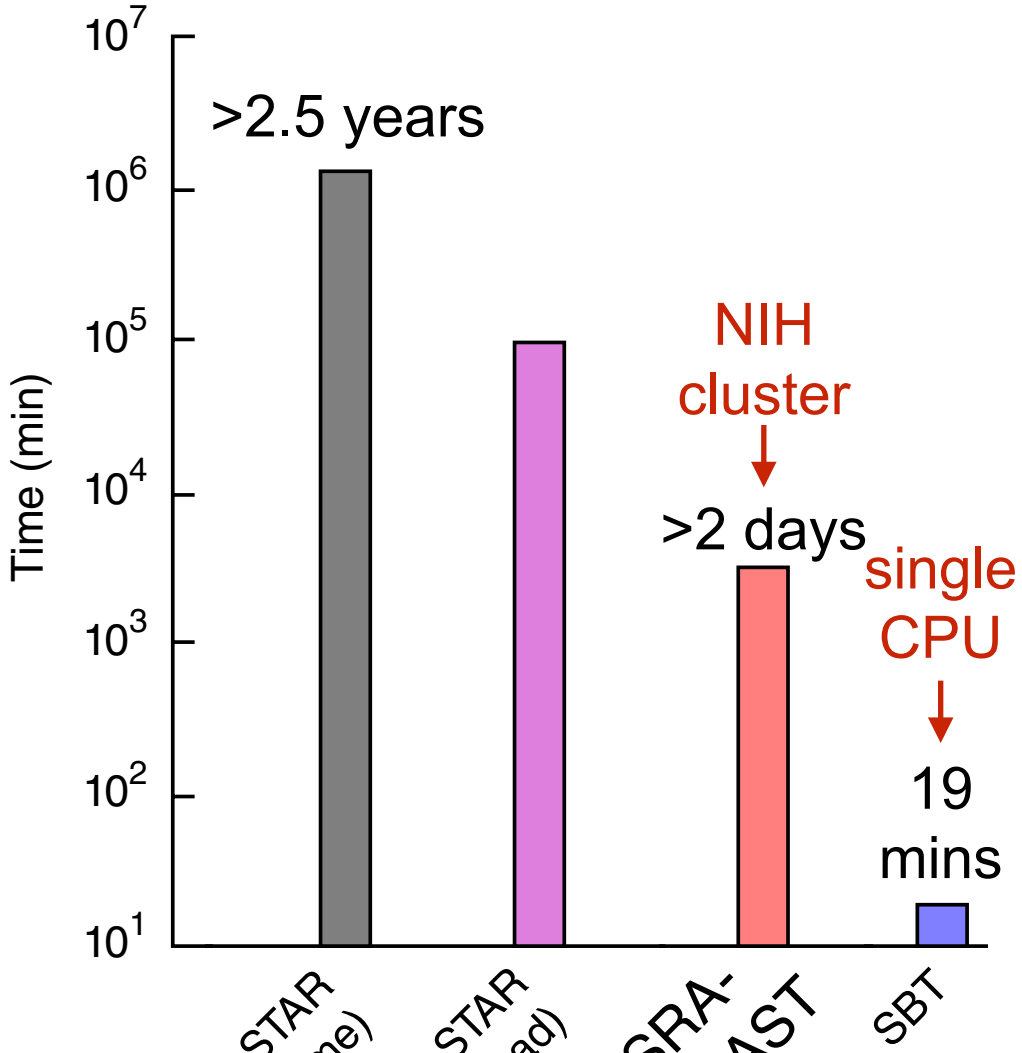
Justify appropriate algorithms for data science problems

Decompose problem into supporting data structures

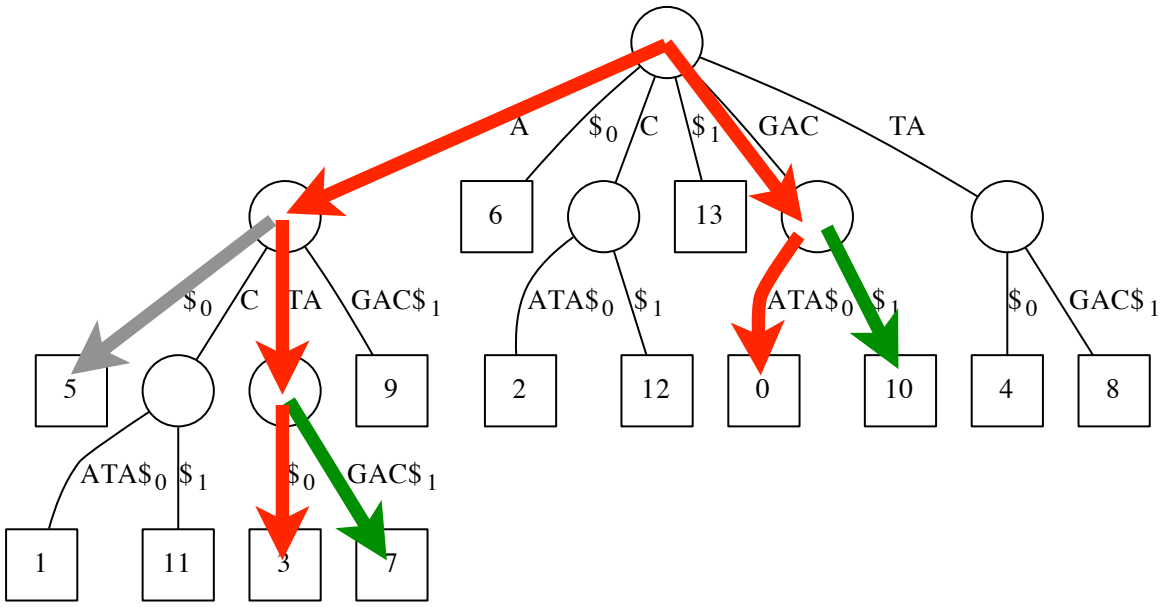
Analyze efficiency of implementation choices

Why should you care?

Justify appropriate algorithms for data science problems



Find all overlap: GACATA vs ATAGAC



CS 225 vs CS 277

The prerequisite requirements for CS 277 are very different

The learning goals and content are also very different.

CS 225 is necessary to enroll in many upper-level CS classes

Course Webpage

https://courses.grainger.illinois.edu/cs277/sp2024/

All course information and links can be found here!

- Course Schedule and Lecture Material
- Assignment links and descriptions
- Piazza links and Office Hours
- Syllabus

In-Lecture Course Expectations

Attendance is encouraged but not mandatory

Ask questions!

Participate in class exercises / labs

Out-of-lecture Course Expectations

Weekly assessments

Lab assignments are published **Friday** and due **Monday @ 11:59 PM**

Mini-projects deadlines are published with each project (~3 weeks)

Watch recorded lectures (if you missed in-person)

All lectures are published on Mediaspace:

https://mediaspace.illinois.edu/channel/CS+277/225216063

Grading

Category	Contribution	Notes
Mini-Projects	300	75 points each
Labs	300	25 points each
Exams	300	75 points each
Final	100	+ 1 retake exam

Points	Grade
900	A-
800	B-
700	C-
600	D-
	F

Mental Health

This class should be low-stress, medium work-load.

UIUC offers a variety of confidential services:

Counseling Center: 217-333-3704

610 East John Street Champaign, IL 61820

McKinley Health Center: 217-333-2700

1109 South Lincoln Avenue, Urbana, Illinois 61801

Diversity, Equity, and Inclusion

"If you witness or experience racism, discrimination, micro-aggressions, or other offensive behavior, you are encouraged to bring this to the attention of…"

Course CAs

Faculty

Campus Belonging Office (Link)

The Office of Student Conflict Resolution (Link)

CS CARES (Link)

Class structure is under development!

Class size tripled from last iteration, course description changing

Frequent assessment will allow adjustments as needed

Learning Objective: Programming

Navigate, organize, and run moderately complex Python projects

Three sub-goals to be a better programmer:

1) Building up your 'programming toolbox'

2) Thinking carefully about a problem

3) Choosing the right tool for the job

Programming Toolbox: Variables

What is a variable in Python?

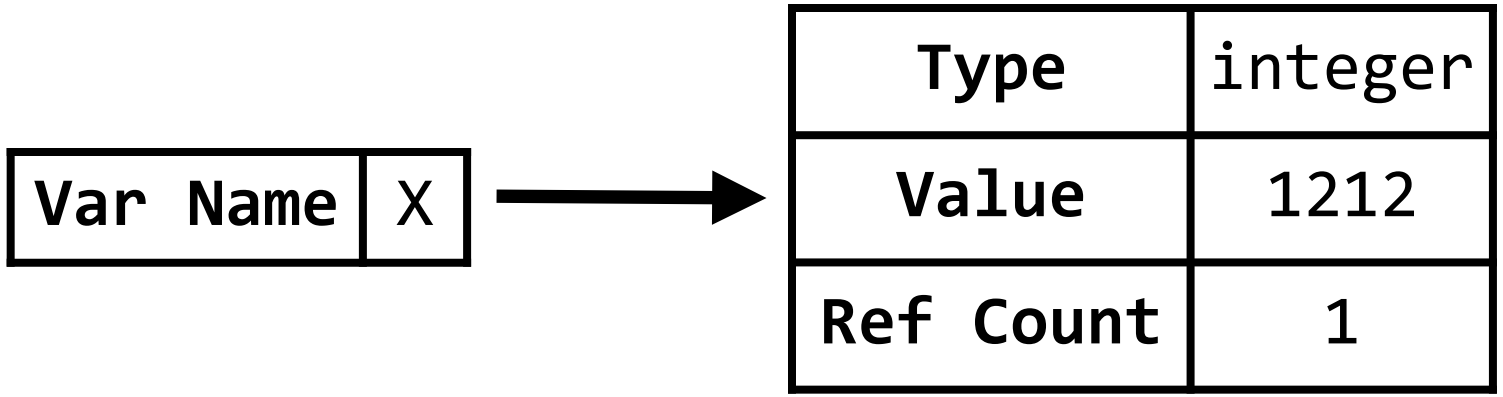
```
a="3"
b=3
c=3.0
d=True

print(a + b)

print("3 + 3”)

print(b + c)

print(c + d)

print(d)

print(d - d)
```

What information is necessary to define a variable?

Programming Toolbox: Variables

Everything in Python is an **object**

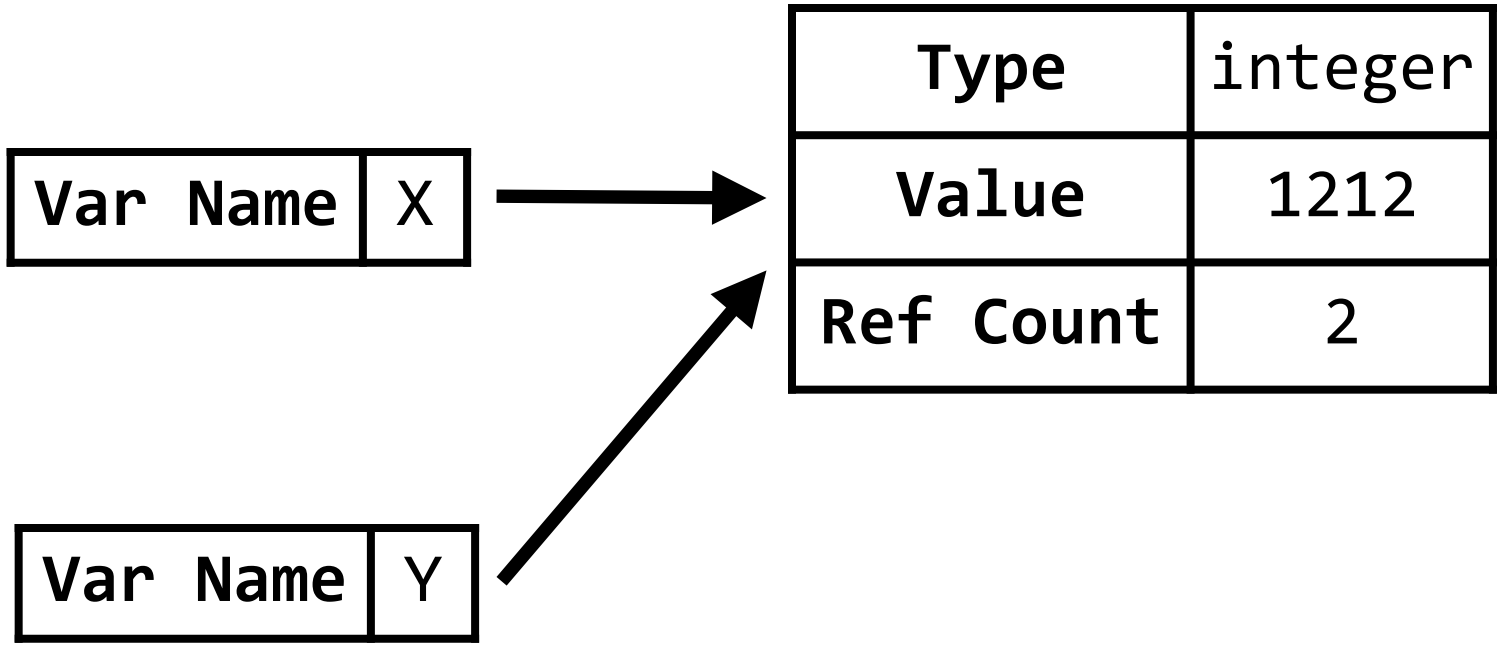
```
1   x = 1212
2
3
4
5
6
7
8
9
10
```



Programming Toolbox: Variables

Everything in Python is an **object**

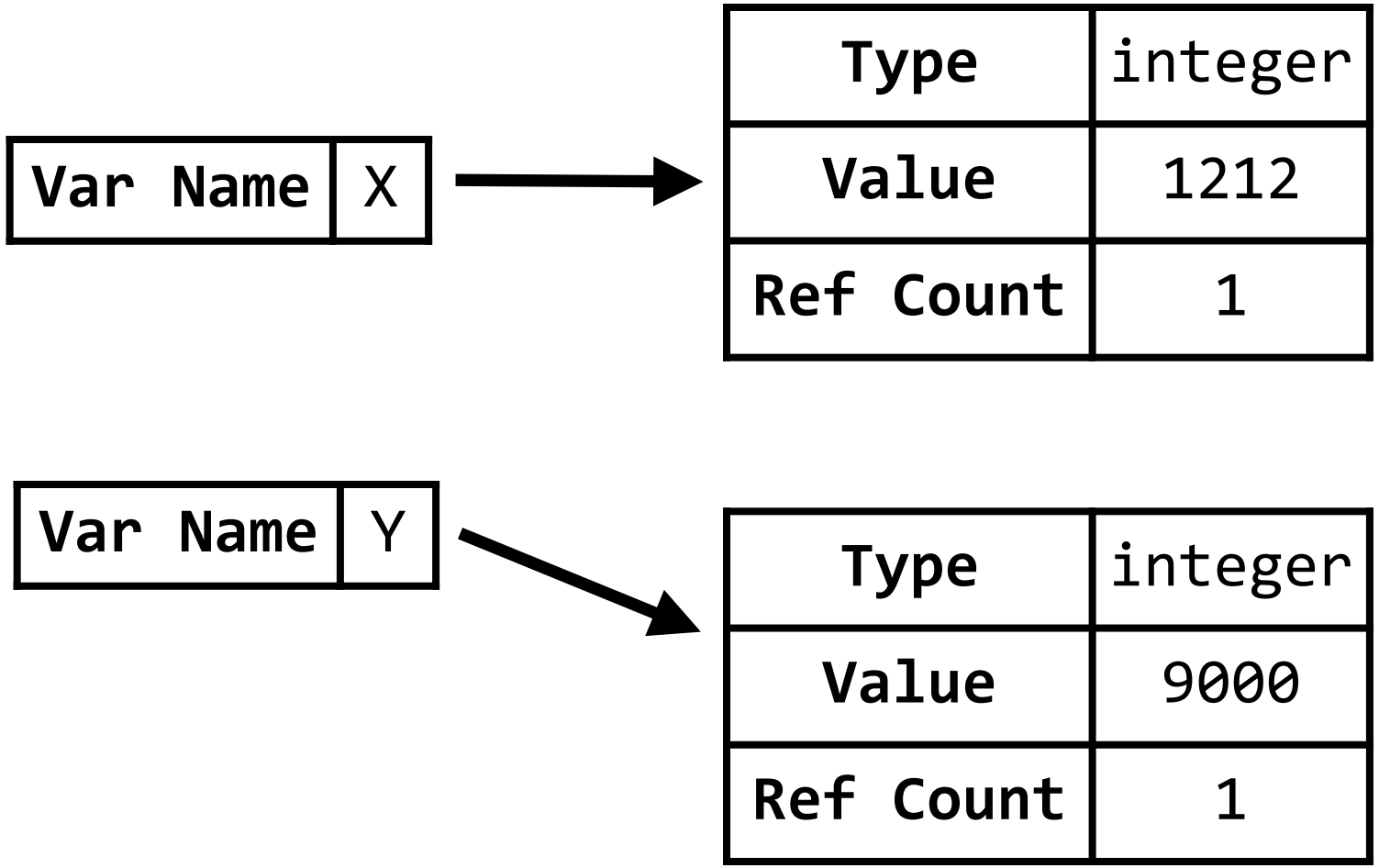
```
1   x = 1212
2
3   Y = x
4
5
6
7
8
9
10
```



Programming Toolbox: Variables

Everything in Python is an **object**

```
1   X = 1212
2
3   Y = X
4
5   Y = 9000
6
7
8
9
10
```



Programming Toolbox: Variables

Everything in Python is an **object**

```
1   x = 1212
2
3   y = x
4
5   y = 9000
6
7   x = 12
8
9
10
```

Type	integer
Value	1212
Ref Count	0

Var Name	X

Type	integer
Value	12
Ref Count	1

Var Name	Y

Type	integer
Value	9000
Ref Count	1

Programming Choice: Data Type

Python has many built-in data types:

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

```
string = "Hello World”

intv = 1

floatv = 1.0

listv = [1, 2.0, “:)"]

dictionary = {"Key" : “Value"}

boolean = True

setv = {1, 3, 5, 7, 9}

```

Programming Choice: Data Type

Which of the following will result in a variable **x** having the value **0.5**?

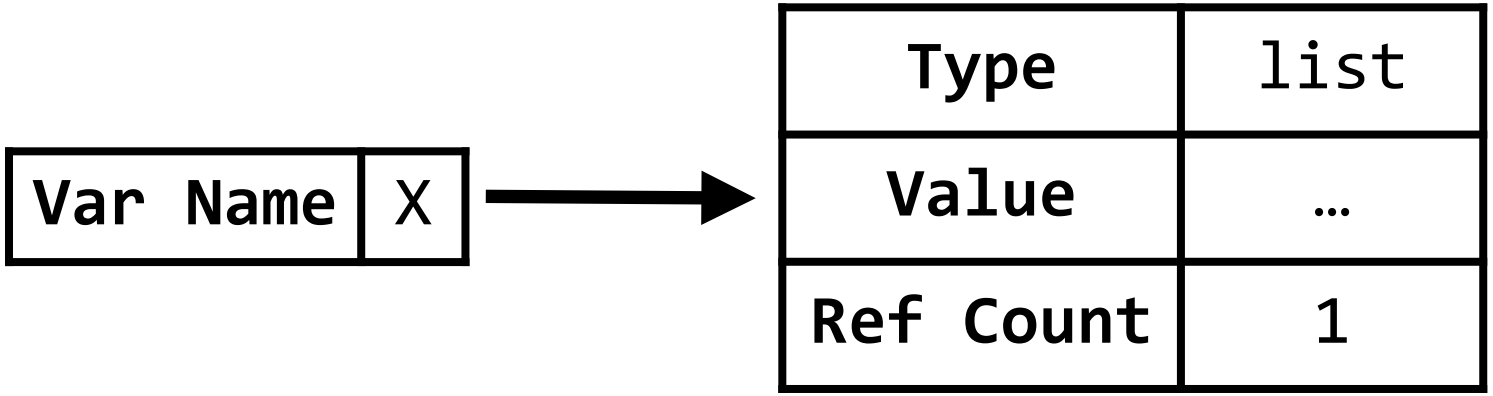
```
# ** A **
x = 1 / 2

# ** B **
0.5

# ** C **
y = 1.0
x = y / 2

# ** D **
x == 0.5

# ** E **
2*x = 1

# ** F **
0.5 = x
```

Of the valid solutions, which do we like the most?

Programming Choice: Data Type

Some objects are **mutable** — we can change their values after creation

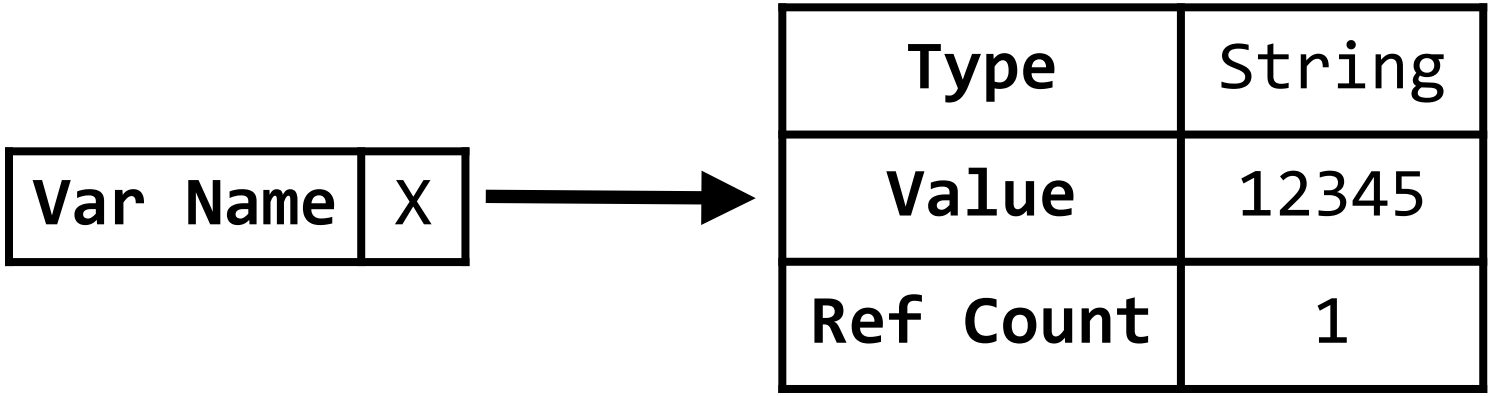
```
1   x = [1,2,3,4,5]
2
3   print(id(x))
4
5   x[2]=0
6
7
8   print(id(x))
9
10  print(x)
```



Programming Choice: Data Type

Some objects are **immutable** — you have to make a new object

```
1   x = "12345"
2
3   print(id(x))
4
5   x[2]=0
6
7
8   print(id(x))
9
10  print(x)
```



Why do we care?

Imagine you have two datasets:

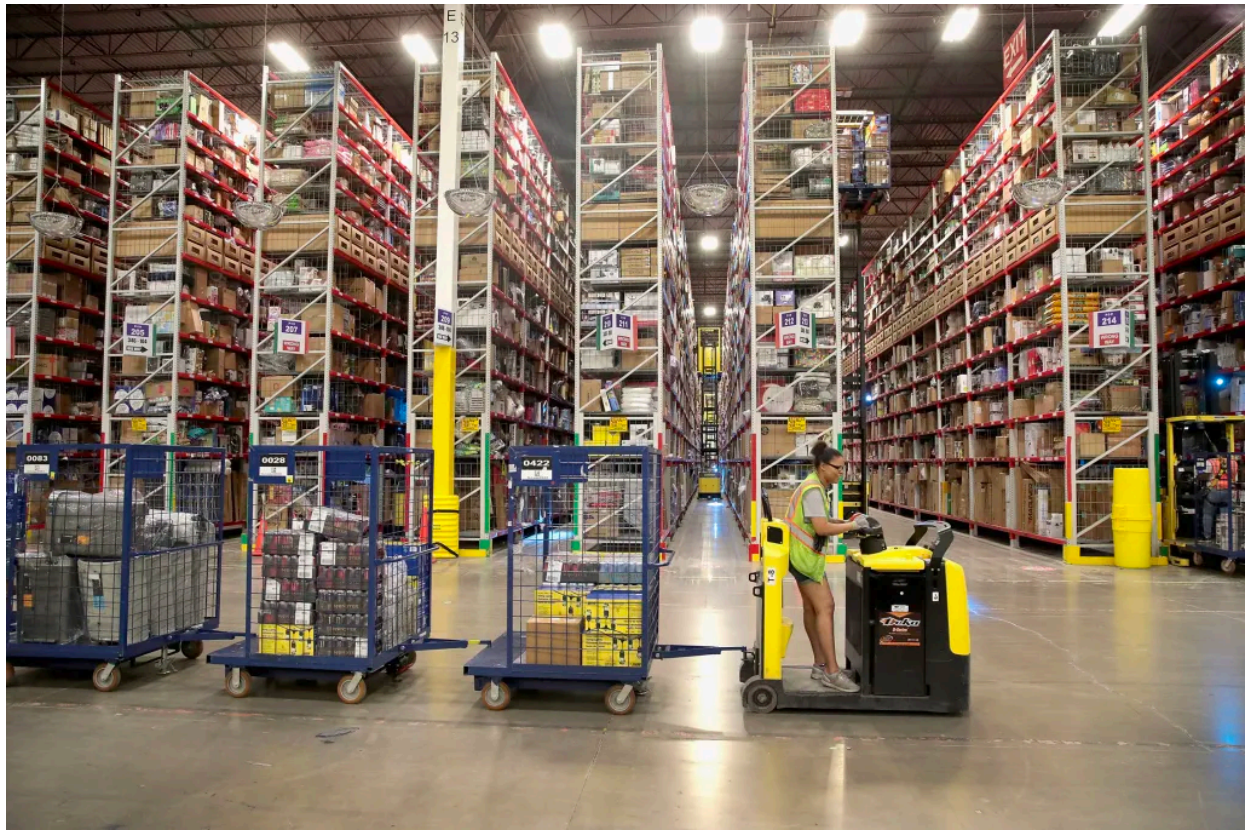
Dataset 1 is *very* large but fixed in size.

Dataset 2 starts off small but grows to an unknown size.

Would you rather have a **mutable** or **immutable** variable?

What do we care about when writing code?

What do we care about when writing code?



Tying it all back together…

For most simple programs or small datasets, efficiency doesn't really matter

But this will not always be true — especially in the data sciences!

```
def type1(strList):
    out = ''
    for s in strList:
        out += s
    return out

def type2(strList):
    return ''.join(strList)
```

Programming Toolbox: Conditionals

Conditional statements control what blocks of code get run

```
num = 20

if num in [0,1,2,3,4]:
    print("Top 5!”)

elif num > 10:
    print("num too large!”)

elif num > 15:
    print("will this ever get called?”)

else:
    print(num)
```

Programming Toolbox: Conditionals

Whats the difference?

```
num = 2

if num >= 1:
    print("A")
elif num >= 2:
    print("B")
elif num >= 3:
    print("C")
```

```
num = 2

if num >= 1:
    print("D")
if num >= 2:
    print("E")
if num >= 3:
    print("F")
```

Programming Toolbox: Loops

We are often tasked with processing every item in a dataset.

We use loops to simplify our code structure.

```
for i in range(3):
    print(i)




count = 0
while(count <= 2):
    print(count)
    count+=1
```

For Loop:

While Loop:

Programming Toolbox: Loops

There are a number of useful keywords for writing loops

```
count = 0

while(True):
    if count % 2 == 0:
        count+=1
    else:
        pass

    if count > 10:
        break
    else:
        count+=1
        continue
        count+=1
    print('count: {}'.format(count))

print('count: {}'.format(count))
```

Pass:

Break:

Continue:

What does this code print?

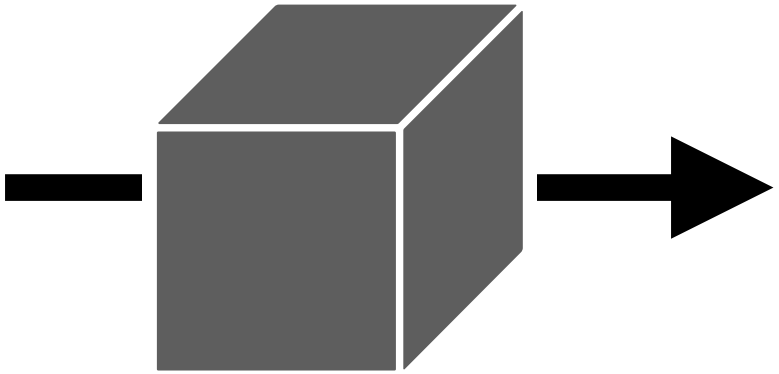
```
count = 0

while(True):
    if count % 2 == 0:
        count+=1
    else:
        pass

    if count > 10:
        break
    else:
        count+=1
        continue
        count+=1
    print('count: {}'.format(count))

print('count: {}'.format(count))

```

Programming Toolbox: Functions

Functions are the building blocks of programming

Input

Output



```
def type1(strList):
    out = ''
    for s in strList:
        out += s
    return out

def type2(strList):
    return ''.join(strList)
```

```
def mystery(inValue):
    return inValue + inValue
```

Programming Toolbox: Functions

You should always document the intended input and output.

```
# INPUT:
# A string (checkin)
# A string (checkout)
# OUTPUT:
# A float storing the number of minutes between checkin and checkout time.
def getTotalTime(checkin, checkout):
    return inValue + inValue
```

```
def mystery(inValue):
    return inValue + inValue
```

Programming Toolbox: Functions

Immutable variables created in a function have **local scope**

Mutable variables can be modified by functions

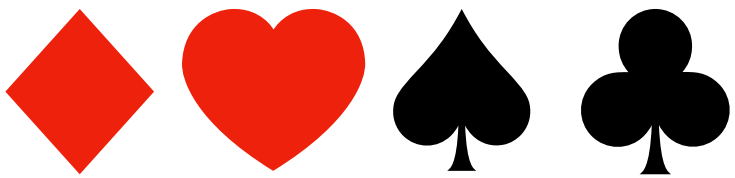
```
def scopeTest(inNum, inString, inList):
    inNum = 3

    inString+="And After!"


    inList.pop(-1)
    inList.append(5)

x = 2
y = "Before! "
z = [1,2,3,4]

scopeTest(x,y,z)

print(x)
print(y)
print(z)
```

Programming Toolbox: Functions

Functions are **objects** (like everything in Python)

```
# INPUT:
# Three integers (a, b, c)
# An optional function (f)
# OUTPUT:
# If f exists, return output of f(a, b, c). Else return defaultF(a,b,c)
def wrapperFunction(a, b, c, f=None):
    if f == None:
        return defaultF(a,b,c)
    else:
        return f(a,b,c)


if __name__ == '__main__':
    wrapperFunction(5,3,2, add)


    wrapperFunction(1,1,1, multiply)

```

In-Class Exercise: Slot Machine



Let's program the output of a slot machine!

$500 = Four matching symbols

$100 = Four of a color

$50 = Three matching symbols in a row

$10 = One of each symbol

First Lab Friday!

Bring your laptop (first part will be going over installation instructions)

Lab will focus on how to break down a programming problem

Friday Foreshadowing: getTotalTime()

Given **HH:MM:SS** format, I want to know the exact difference between start and stop times in minutes. How would we approach this problem?