CS 5004

LECTURE 1: OVERVIEW

CLASSES & OBJECTS

KEITH BAGLEY

Northeastern University
Khoury College of
Computer Sciences

440 Huntington Avenue ▪ 202 West Village H ▪ Boston, MA 02115 ▪ T 617.373.2462 ▪ khoury.northeastern.edu

AGENDA

- Welcome Back!
  - Intro
- Big Ideas
- Course Themes
- Course Logistics
  - Review of the Syllabus, CANVAS, Office Hours & Other Course Items
- Q & A

REMEMBER THIS?

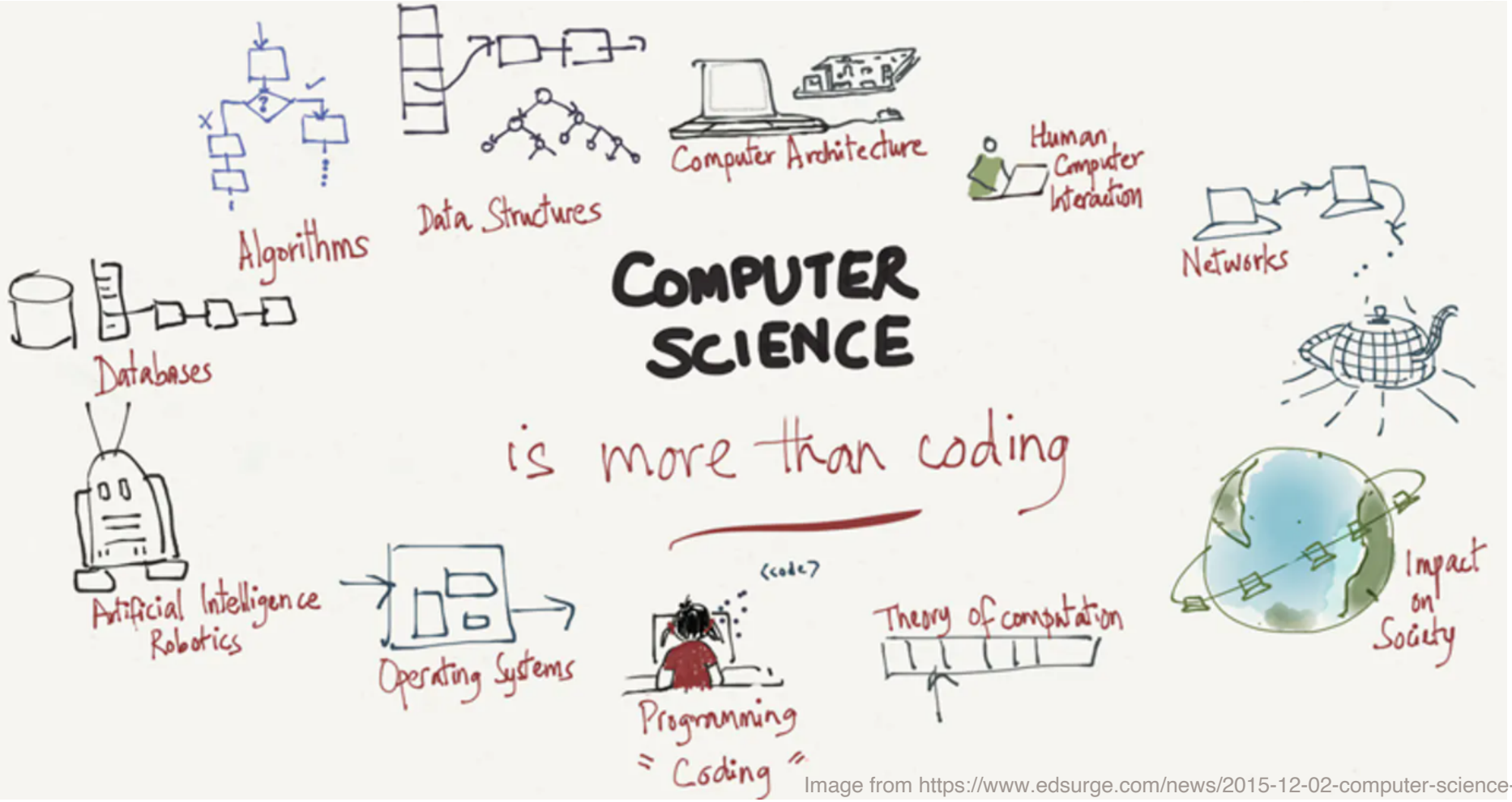


Image from https://www.edsurge.com/news/2015-12-02-computer-science-goes-beyond-coding
Courtesy of Sheena Vaidyanathan

A BIT OF ART & SCIENCE

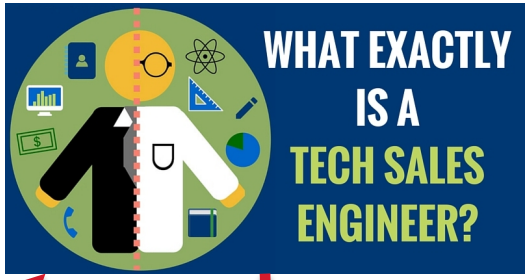
WHO AM I?

- Keith Bagley, Ph.D., D.Min.
- Associate Clinical Professor, Director – MS ALIGN,

Plus…
- Industry
- Husband
- Father
- Multi-vocational
- Pet owner
- Community Active
- Son!

I'm more than a title @ NU
So are you!
Embrace your holistic self

FOR THOSE WHO DON’T KNOW ME



NEW NORMAL – ENDURING TIPS FOR SUCCESS

- COVID-19 introduced challenges & opportunities
  - “New Normal” – Online by necessity, not by design
  - Opportunity to re-evaluate some content delivery
  - Challenges “staying engaged” with studies & peers
- Remember who you are and why you’re here.
  Do not let external circumstances alter your internal compass!
- Truism: The only thing “constant” is the fact that things will change
- Touchdowns are scored one play at a time. Move forward every day 🏈
  - Sorry Futbol fans…Goals are scored one kick at a time? ☺ ⚽
- Reach out for help early & often

WARM-UP

- What is your “nemesis food”?

(A) Candy

(B) Burgers

(C) Salad

Yes!

(D) Yes

WARM-UP

- Where would you like to go on vacation after the pandemic?

(A) Hawaii

(B) Carribbean

(C) Florida

(D) Yes

WARM-UP

- Are you excited about CS 5004?

(A) Yes

(B) Heck Yes

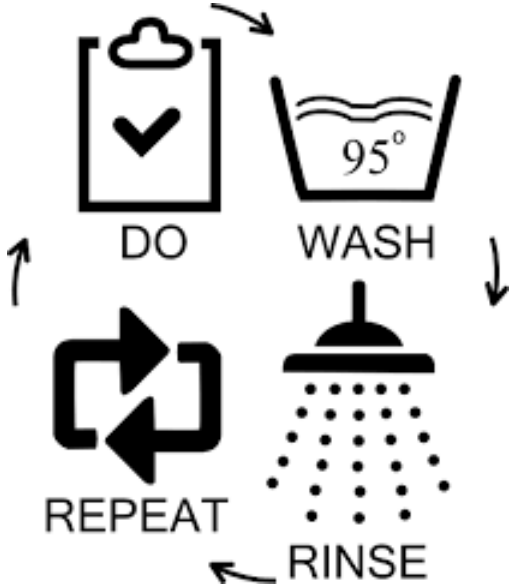
(C) Absolutely Yes

(D) Are you kidding me? YES!

WHAT WE DO - PLAN YOUR WEEK!

- Individual quizzes should be completed by the individual. No collaboration
- Labs can be done in small groups. No penalty for collaboration
- Team-based quizzes are designed to be taken as a team (collaboratively).
- Overview of the week:

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
	LAB & Individual Quiz	Lecture (Clark) Team-based Quiz (Clark's section) Lab due (11pm EST) Lab self-eval due (11:59pm EST)	Lecture (Keith) Team-based Quiz (Keith's section)	Homework Assigned / Homework Due	Next week's Module posted Homework self-eval due	



HOMEWORK

- Individual for the first 2/3 of the course – no code sharing
- First few assignments are “prescriptive”; last few assignments are “open ended” with respect to what solution approaches you can use
- Grading
  - Automated Style checker
  - Solution correctness via Server-side tests
  - Self-evaluation
  - Manual grading

- Note: Homework & Lab assignments can be submitted as many times as you want before the deadline. **Self-evals can only be submitted ONCE!**
  - If you submit your self-eval, the system will NOT allow you to resubmit your homework. BE SURE your homework is submitted prior to submitting your self-eval

LATE POLICY

- FOUR (4) No-questions asked late days automatically managed by the Handin Server
  - No need to email professor for permission
  - Only ONE (1) late day can be used on any given assignment
  - Late days are used on a per-assignment basis (e.g.: a late homework uses 1 “late day”; a late self-eval for said homework uses another “late day”)

FLIPPED WORK

- You have more control over your learning experience. You will need to come to each class prepared
- **Pre-class work**
  - Module readings finished & and videos watched. In addition, you will be required to complete an individual quiz to the best of your ability.
- **In-class work**
  - Be engaged in this course! The more you engage, the more you'll benefit
  - Participate in class discussions, breakout sessions & coding exercises
  - Respect your classmates' right to learn, and to refrain from disruptive

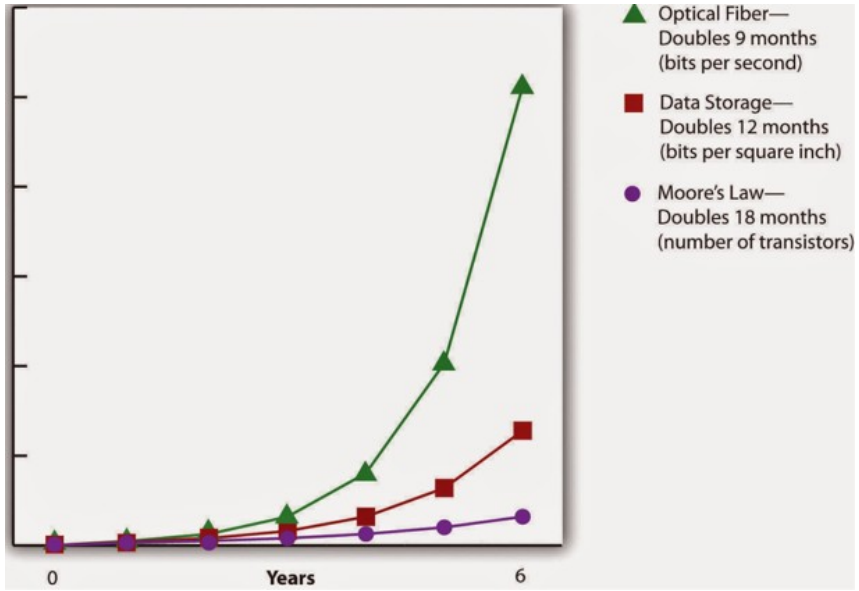
WHAT'S DIFFERENT TODAY?

- Note: **Most lectures will be in "flipped format"**
  - This is important! I'll do <u>some</u> "standard" lecturing, but much of our engagement in this section will be facilitated Q&A, working sessions, dialogue and concept tours
    - If you prefer more "traditional" lectures, Prof Clark does that on Tuesday evenings, 6pm
  - Today I'll be talking & presenting more than usual while we get ramped up
  - Most "lecture days" will be more facilitated reviews of the material you've already seen

BIG IDEAS

- Software & Data are everywhere!
  - Consumption & production rate is increasing exponentially

MOORE'S LAW VS. DATA



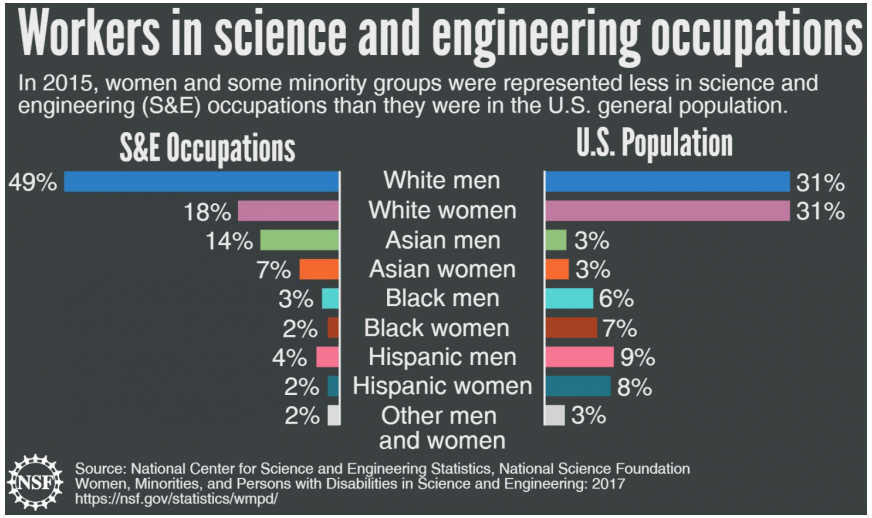
Future estimates...

Just one autonomous car will use 4,000 GB of data/day

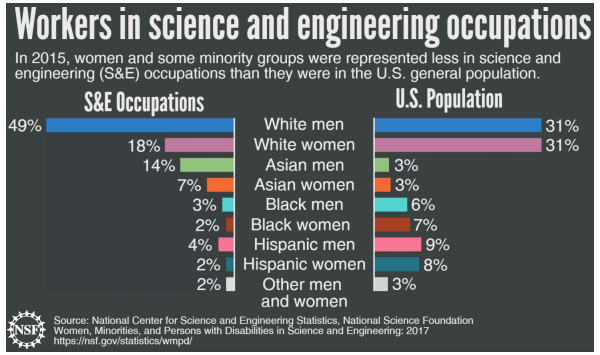
Self-driving cars will soon create significantly more data than people—3 billion people's worth of data, according to Intel

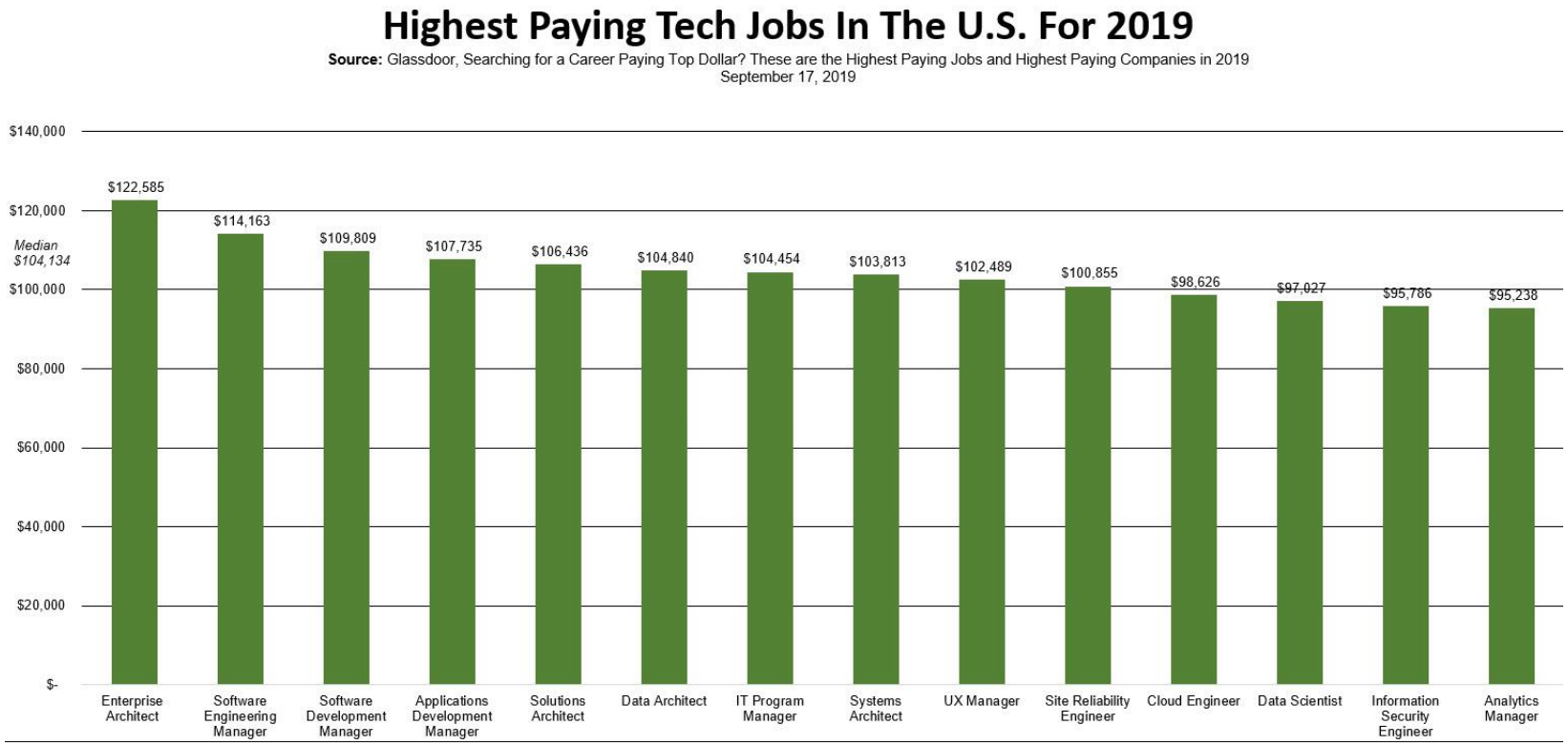
BIG IDEAS

- Broadening Participation
  - Digital divide
  - Some groups underrepresented in CS and Tech
    - Those who control the code control the code biases & behavior
    - Tech is ONE avenue for wealth generation



BIG IDEAS: CS FOR EVERYONE





COURSE THEMES

- Abstraction
  - Ignoring certain details to focus on higher aspects (verb)
  - The product of the process of abstraction (noun)
- Object-Oriented vs. Procedural
  - We did mostly procedural in CS5001, with a touch of OO
- Testing & Debugging
- Java language
  - Similar to CS5001 & Python, we're using Java as the vehicle to learn concepts
- Design Patterns

LET'S GET CRACKIN'

WHERE ARE WE COMING FROM?

- If you know Python, or are coming to CS 5004 from CS 5001, you're familiar with:
  - a dynamically typed language
  - a language that doesn't require you to "look under the hood" often
  - a language that has nice features that handle mundane "housekeeping"
  - a hybrid language and system that supports multiple programming paradigms:
    - Procedural, Functional, Object-Oriented
  - a language that sounds dangerous, but is actually from a comedy troupe

  Python

WHERE ARE WE GOING?

- CS 5004 will immerse you in
  - a (more) statically-typed language
  - a language that requires a bit more attention to details
  - a language that has powerful features and that handles *some* mundane "housekeeping"
  - a fully object-oriented language & system
  - a language that sounds like a drink, but originally named after a tree (Oak)

  Java

ANCIENT EGYPT ANALOGY

Python

“Can it scale? Probably, but...”
“Can it feed you grapes? Yes!”

Java

”Can it scale? Absolutely!”
“Can it feed you grapes?
Uh. It just squished your grape”

PROCEDURAL APPROACH WITH FUNCTIONS

- Using our old friend Python to help us:

```python
def calc_area(choice):
    ''' function do_area
        Input: choice - the type of shape selected by the user
        Returns: area of a shape, depending on the type chosen
        Does: Asks the user for shape's dimensions to cacluate area
    '''
    if choice == 'S':
        length = float(input('Enter the square length: '))
        return round((length * length),2)
    elif choice == 'C':
        radius = float(input('Enter the circle radius: '))
        return round((PI * radius**2),2)
    elif choice == 'T':
        base = float(input('Enter the triangle base: '))
        height = float(input('Enter the triangle height: '))
        return round((0.5 * base * height),2)
    return 0
```

```python
def main():
    answer = ''
    while answer != 'Q':
        question = ('Select the Shape you want:\n' +
        ' S: Square\n C: Circle\n T: Triangle\n' +
        ' Q: Quit?\n Your answer: ')
        answer = menu(question)
        if answer == 'Invalid Choice':
            print(answer, '...please choose again')
        elif answer != 'Q':
            print('Area = ', calc_area(answer))
        else:
            print('Thanks for using my Shape program!')

main()
```

PROCEDURAL APPROACH: A PYTHON VIEW

- Using our old friend Python to help us:

```
def calc_area(choice):
    ''' function do_area
        Input: choice - the type of shape selected by the user
        Returns: area of a shape, depending on the type chosen
        Does: Asks the user for shape's dimensions to cacluate area
    '''
    if choice == 'S':
        length = float(input('Enter the square length: '))
        return round((length * length),2)
    elif choice == 'C':
        radius = float(input('Enter the circle radius: '))
        return round((PI * radius**2),2)
    elif choice == 'T':
        base = float(input('Enter the triangle base: '))
        height = float(input('Enter the triangle height: '))
        return round((0.5 * base * height),2)
    return 0
```

Functions and data are separate

I pass choice into calc_area() as outside data

Conditional code to determine data

”if” statement to determine what to do with the data

Handing back the data to a client

The function doesn’t “own” the data so control passes to someone else who might modify it, depending on the type of data it is

PROCEDURAL APPROACH

- Nothing is inherently wrong with a procedural approach; it's been used in computing for decades
- Some trade-off's with other programming paradigms
  - e.g. not "owning" the data, possible 'ripple effect' as conditionals are used to trigger behavior, etc.
- If you are familiar with Python, it is a hybrid language that allows multiple paradigms (including procedural) to be used

OBJECT ORIENTED APPROACH

- What if we took a different approach where:
  - The data and the functions that operate on that data were grouped together?
  - We could build our own abstractions for our programs and extend the system?
  - We could send conceptually similar “messages” to instances of these abstractions, and have them respond appropriately without needing a bunch of “if” statements?
- We would call that an **object-oriented** approach

OBJECT ORIENTED APPROACH

- Using our old friend Python again, since it supports OO too!

```python
PI = 3.1415

'''
    Class: Square.
    It knows how to draw itself, calculate its area, compare to other
    Shapes (for equality) and calculate its perimeter. Some code
    duplication in these classes since we're not using inheritance, but
    since Python is a dynamically typed language we don't need inheritance
    to implement protocol/interfaces
'''
class Square:
    def __init__(self, length):
        self.length = length
        self.name = 'Square'
    def perimeter(self):
        return self.length * 4
    def area(self):
        return round(self.length**2,2)
    def __str__(self):
        return self.name + " with length of " + str(self.length)
    def __eq__(self, other):
        if self.area() == other.area():
            return True
        return False


def main():
    # construct random shapes and load them into a list.
    print("Hi there! Let's make some Shapes!")
    shapes = []
    for i in range(10):
        shapes.append(shape_factory())
    for each in shapes:
        print("{} : has an area of {}".format(each, each.area()))
        print()
```

Functions & data are grouped

Methods and attributes are owned by the class

Better control of data access**

Class designer determines what data to share and expose**

No conditional code to determine data type

Objects know what they are, and what data they have

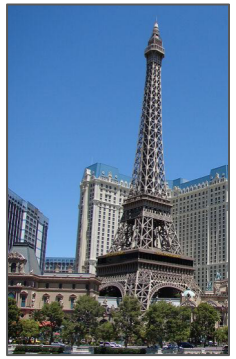
SOME TERMINOLOGY

- **Class**: a blueprint for how to make an object. Classes describe the "thing" (abstraction) we're talking about – its:
  - **attributes** (data it owns/knows),
  - **operations** (behavior, what it knows how to do), and
  - **state and relationships** to other things
- **Object**: an "instance" (or one of the "things") from the class
  - Objects have attributes, operations and unique identity

blueprint

Paris

Las Vegas



Other (almost) synonyms you might hear:
Instance variables -> attributes
Methods or "member functions -> operations

SOME TERMINOLOGY

- **Encapsulation**: an object's data is packaged together with its corresponding operations
- **Information Hiding**: uses encapsulation to emphasizes the separation of external "visible" properties of an object from internal hidden properties, and restrict access by enforcing well-defined interfaces

Encapsulation

Information Hiding

OO CONCEPTS ARE GENERAL ACROSS LANGUAGES

- Depending on the language, some practitioners prefer one of the synonyms
  - E.g. “member function” rather than method
- Not all languages provide facilities for every concept
  - E.g. Python provides encapsulation, but only rudimentary (if any) information hiding

```
def main():
    # construct shapes and load them into a list. No 'if' statements
    # required here

    shapes = []
    for i in range(10):
        shapes.append(shape_factory())

    for each_shape in shapes:
        each_shape.introduce_yourself()
        print('--')
        print(each_shape.name) # encapsulation, not info hiding!
```

Not information hiding!

OBJECT ORIENTED APPROACH: JAVA

- Let’s visit our new friend (or frenemy)

OBJECT ORIENTED APPROACH: JAVA

Welcome Guide | Square.java

```java
public class Square implements Shape {
  private String name;
  private double length;

  /**
   * Constructor for the Square class. Create an instance and specify its length and set the name
   *
   * @param length  the length of the square's side
   */

  public Square(double length) {
    this.name = "Square";
    this.length = length;
  }

  /**
   * getName(): Answer the name. Here primarily to show the difference w/ the Python version
   * @return the name
   */

  public String getName() {
    return name;
  }

  /**
   * getArea(): Answer the area of the square
   */

  public double getArea() {
    return length * length;
  }
```

Encapsulation + Information Hiding

Notice the **public** & **private** access specifiers

Javadoc comments similar to PyDoc

Also: annotations & Javadoc tags w/ @

Same idea of objects “owning” their stuff

Python uses **self**; Java uses **this**

No conditional code to determine data type

Objects know what they are, and what data they have

OTHER JAVA ELEMENTS

WE'LL COVER SOME OF THIS IN MORE DETAIL LATER

Welcome Guide | Square.java

```java
public class Square implements Shape {
  private String name;
  private double length;

  /**
   * Constructor for the Square class. Create an instance and specify its length and set the name
   *
   * @param length  the length of the square's side
   */

  public Square(double length) {
    this.name = "Square";
    this.length = length;
  }

  /**
   * getName(): Answer the name. Here primarily to show the difference w/ the Python version
   * @return the name
   */

  public String getName() {
    return name;
  }

  /**
   * getArea(): Answer the area of the square
   */

  public double getArea() {
    return length * length;
  }
```

{ Curly Braces } instead of tab formatting to designate "blocks"
Lines end with semi-colon ;

Variables **must** be **defined** with a type before being used. Variable types are unchanging

this is like **self**, but <u>not</u> explicitly passed in to methods

Comments

// is similar to # for single-line comments
/** */ is for Javadoc (sort of like ''' ''')
/* is for regular multi-line comments

LET'S DISCUSS

- Object Orientation starts with understanding the objects in your system and asking
  - What do they know?
  - What do they do?

What does an object know?
(data & relationships)

What does an object do?
(behaviors)

LET'S DISCUSS

- Objects should be cohesive & coherent
  - They should do one theme well
  - They should NOT know everything about your system

- Example Circle

WORKING BREAKOUT - GUIDANCE

- You might not have your development environment set up yet, so we'll do some conceptual modeling
- We want to (eventually) create the code to manage Circle objects for a drawing program. We'd like to be able to talk about the color, perimeter and area of circles. And maybe even ask if the 2 or more circles are the "same"
- In your breakout rooms discuss
  - What do we need to design Circles?
    - What do Circle instances know? How do we represent what they "know"?
    - What do Circle instances do?
    - What does "same" mean in this context?

CODE EXAMPLE

- Help me code the Circle
  - Example of some data types, classes, tests

STRETCH BREAK

FAST TRACK FOR MONDAY'S LAB ①

- Write a class Employee that represents a single hourly employee and a class PayCheck that calculates employee pay for the week. Employees know their names (String) the number of hours they worked in a given week, and their pay rate. PayChecks know an employee name, a rate, and the hours worked. PayChecks calculate the weekly pay based on rate * hours if the hours are 40 or less. If the hours worked exceeds 40, the PayCheck applies an overtime rate of 1.5x for all of the hours in excess of 40. Create:
- An Employee constructor that takes a name and pay rate as parameters and sets the instance values appropriately. This constructor should also initialize the hours worked to zero (0) upon instantiation.
- An Employee method addHoursWorked() which takes a parameter (double) and adds the value of that parameter to the current number of hours the Employee has worked this week.

FAST TRACK FOR MONDAY'S LAB ②

- An Employee method resetHoursWorked() that resets the employee's hours worked for the week to zero.
- An Employee method getWeeklyCheck() which, when called, returns a new PayCheck object that is initialized to the current Employee name, rate and hours worked.
- A PayCheck constructor that takes the employee name, rate, and hours worked as parameters and calculates (and stores) the total pay for the week.
- A PayCheck method getTotalPay() that returns the total pay for the week.
- A toString method for BOTH classes, allowing Employee objects to be represented by the employee name, and PayCheck objects by the totalPay (in proper dollars/cents $xxx.yy format).

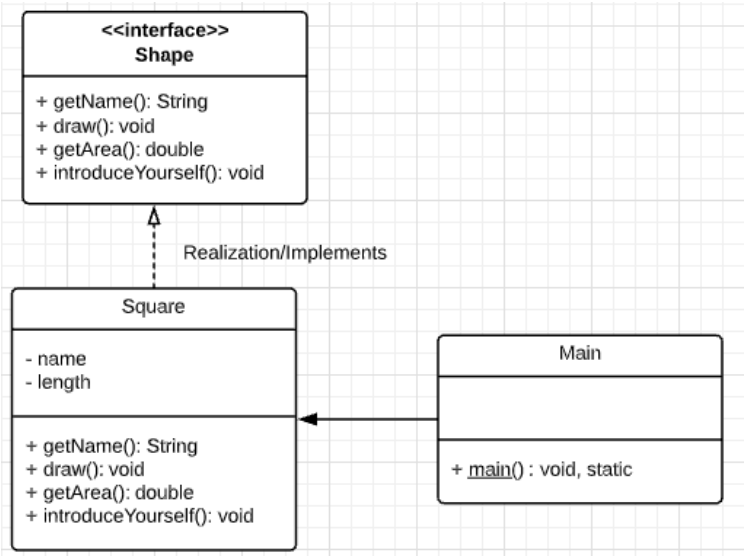
FAST-TRACK BREAKOUT - GUIDANCE

- You might not have your development environment set up yet, so we'll do some conceptual modeling. In your breakout rooms
- Data Types: Look at the employees (and paychecks?)
  - What do we need to store
  - What types of objects should they be?
  - List the methods needed and what the parameters should be.
  - Create the signature for each of those methods

- When we come back, be prepared to share,
  - What assumptions did you make?
  - Types of access?
  - Any initial Test Cases that you thought of

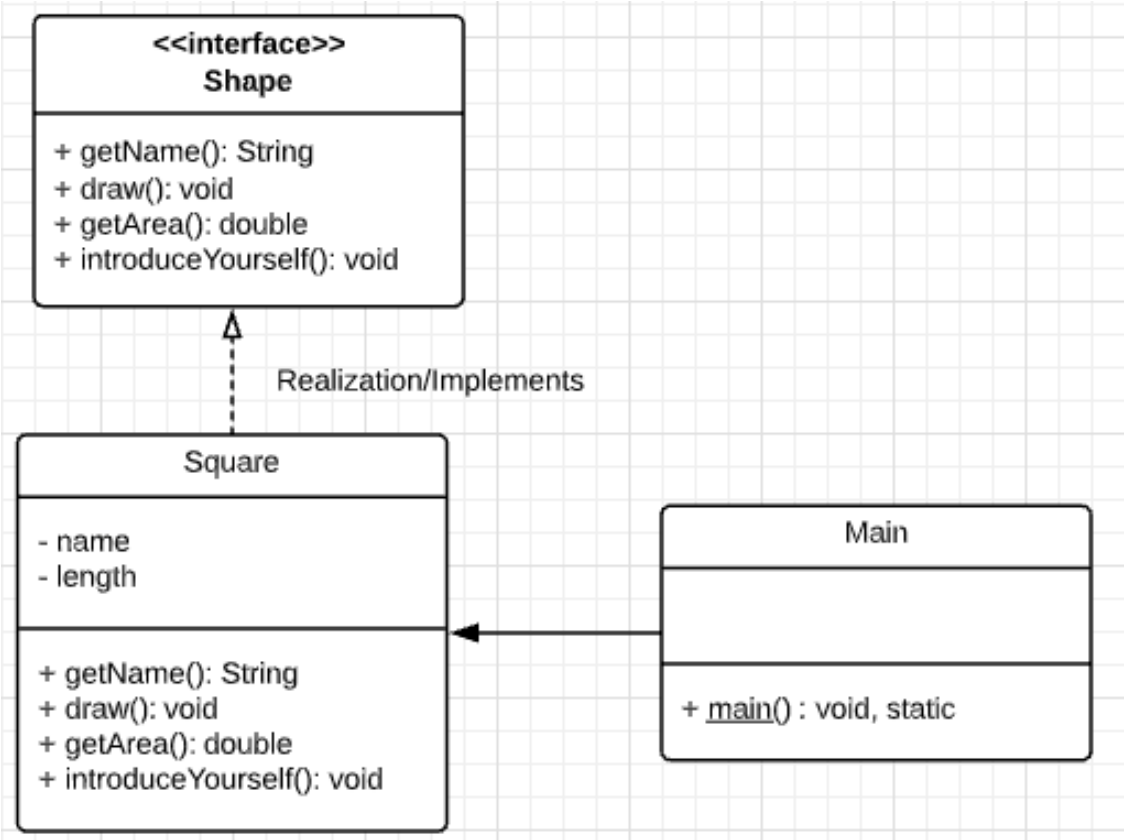
A TOUCH OF UML

- The Unified Modeling Language is a graphical notation with syntax and semantics that raises the abstraction level for specifying and designing systems
  - It can also be used for Model-Driven Development to produce applications via model compilers – a bit on that near the end of this class
  - Acts as a “lingua franca” or common language that cuts across implementations
    - Doesn’t matter if you use Java, Python, C#, etc.
  - Captures both structural and behavioral elements
    - Class diagrams, Interaction Diagrams, State Diagrams, etc



A TOUCH OF UML

- What we have represented is simply **one** view. It's a structural view that shows the "form" of part of our system, and how elements are related
- Other views (behavioral) show the sequence of interactions and message sends between objects, or the state transitions that a single object goes through as events trigger those changes



SUMMARY

- Both Java & Python are OO languages
  - OO concepts are higher-level and transferable across implementation systems. Some languages may not support all features discussed
  - Java is a pure OO language; Python is hybrid/multi-paradigm
  - “better” is in the eye of the beholder
    - Computer scientists & software engineers should pick the best tool for the job

      Or, they’re told what to use by their employer! ☺

COURSE LOGISTICS

- We'll walk through the Canvas syllabus
- Remember to do Module 0 and Module 1 this week.
- Lab 1 is Monday (you've got some concepts worked out from today)
- Module 2 material will be available on Friday. Review that (and take your individual quiz) before lecture next week

Q & A

THANKS!

- Stay safe, be encouraged, & see you next week!