JamCoders: Week 1

Lecture 1A:

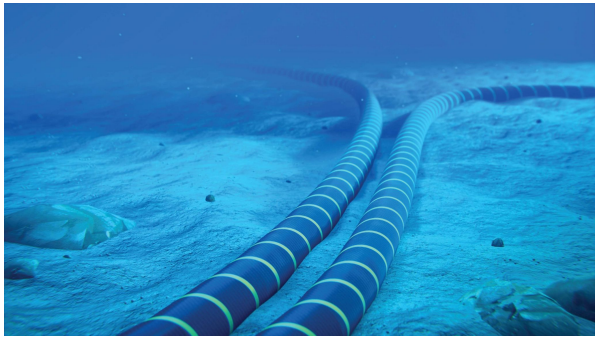
- Intro, CS Demo
- Arithmetic
- Expressions
- Variables

Intro & Logistics

Who am I?

- My name is Alex Krentsel.
- Senior Researcher at Google, designing internet architecture.

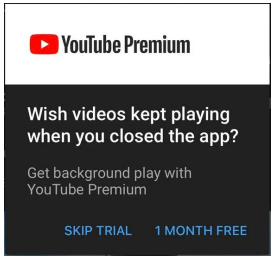
Google



Subsea cable map

Berkeley UNIVERSITY OF CALIFORNIA

Who am I?



- My name is Alex Krentsel.
- Senior Researcher at Google, designing internet architecture.
- Computer Science PhD Student at UC Berkeley
- Previously:
  - Software Engineer at YouTube
  - CS Lecturer at Howard University

Google

Live in California. In my free time, I like to play violin, hike, and cook.

Who are *you?*

What is *this* course?

Course Overview

Goal: build a core foundation of *computational thinking* through studying **data structures** and **algorithms**.

4 weeks of instruction:

- Week 1: Core Python Programming Concepts
- Week 2: Advanced Programming
- Week 3: Algorithms
- Week 4: Data Structures

Lecturers

Alex Krentsel
UC Berkeley/Google

Dr. Timnit Gebru
DAIR Institute

Prof. Jelani Nelson
UC Berkeley

Orr Paradise
UC Berkeley

Who?

- Fantastic teaching team
  - Mix of current students (undergrad, graduate), and industry workers
  - *ask them questions*

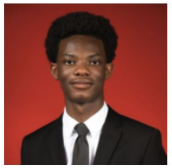
Teaching Assistants


Samuel Bobick
UC Berkeley


Zaria Chen Shui
UWI Mona

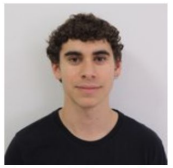

Adrianna Clashing
Campion College


Piyush Goyal
Google


Xavier Henry
Herschel V. Jenkins High


Lydia Ignatova
UC Berkeley


Joy Liu
UC Berkeley


Bruno Monteiro
Federal University of Minas Gerais


Hanna Schlegel
UC Berkeley


Frank Shang
New York University


Manolis Vasilakis
Université Paris-Dauphine

Day Structure

Morning/Afternoon blocks, made up of…

- **Lecture**: big room, one lecturer teaching at the front, demonstrating on the computer, no computers
- **Lab**: smaller room, each student working on their own computer to solve problems, TAs around to help

TIME	SCHEDULE
9:00-10:00	Morning Lecture A
10:00-10:15	Break
10:15-12:15	Morning Lab A
12:15-1:45	Lunch
1:45-2:45	Afternoon Lecture B
2:45-3:00	Break
3:00-5:00	Afternoon Lab B

Week 1 Overview

Learning Objectives for Week 1:

- Learning the *“language of computers”*
  - How to express logic in code to solve problems
  - “Computational thinking”: Code blocks as units of work

- By the end of this week, you’ll know…
  - Core: variables, expressions, operators, conditionals, print, input
  - Types: booleans, ints, floats, strings, lists; casting
  - Loops: for, while, range, nesting
  - Functions: defining, calling, args. vs params, scope, return
  - Misc: mutability, style, errors

Lecture Structure

- Mix of slides and Jupyter Notebook
- Questions throughout
- Interactive!

Let’s begin!

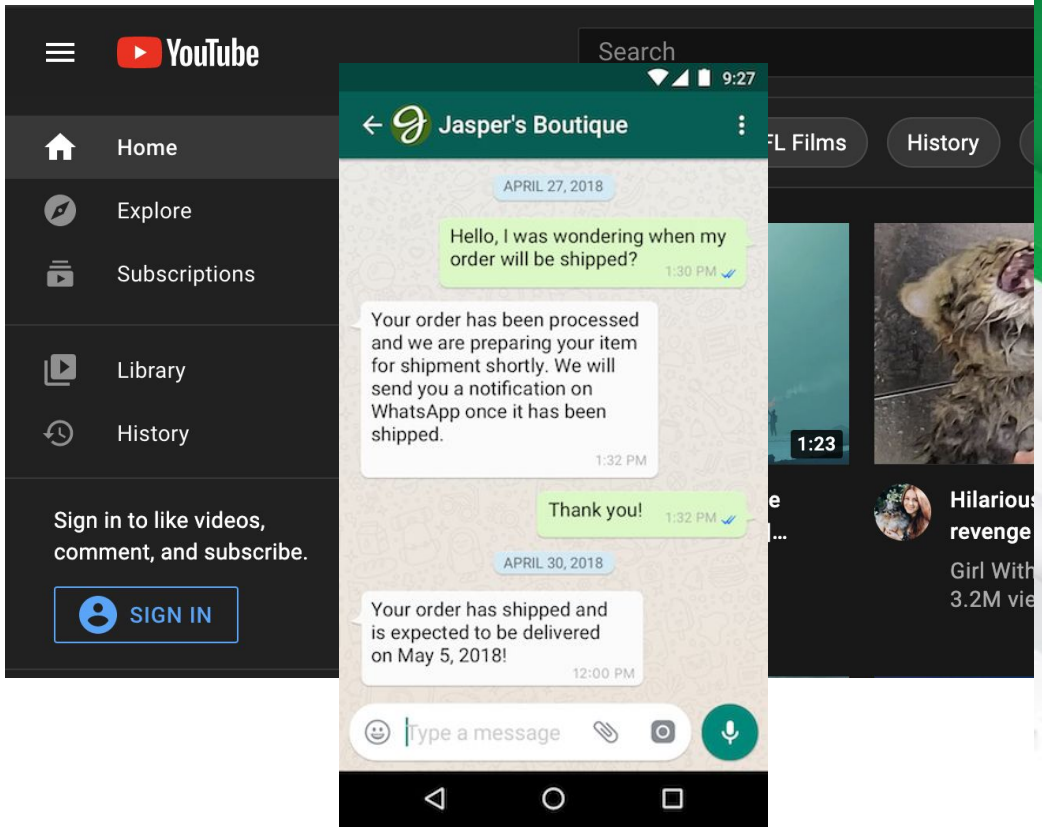
Lecture Goals

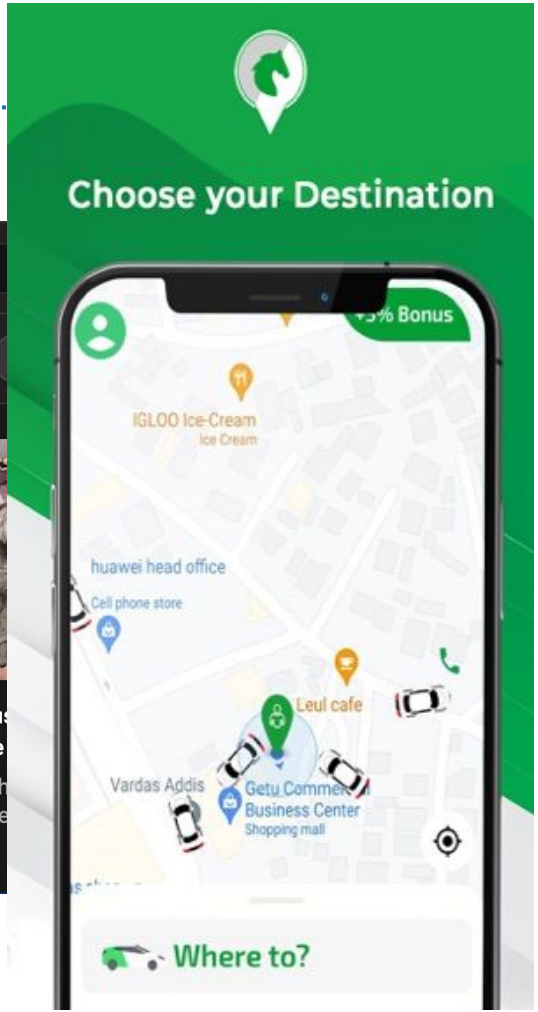
By the end of lecture 1A, you will know…

- Course logistics (just finished)
- Fun: what you can do with CS in 2 weeks
- What is programming and where do we do it?
- Basic building blocks of code (30min)
  - Math Operators
  - Errors
  - Variables
  - Printing
  - Types

Why Study Computer Science?

Modern daily life power by CS.





Why Study Computer Science?

Major driver of progress of our civilization.

Ancient Infant's DNA Reveals New Clues to How the Americas Were Peopled

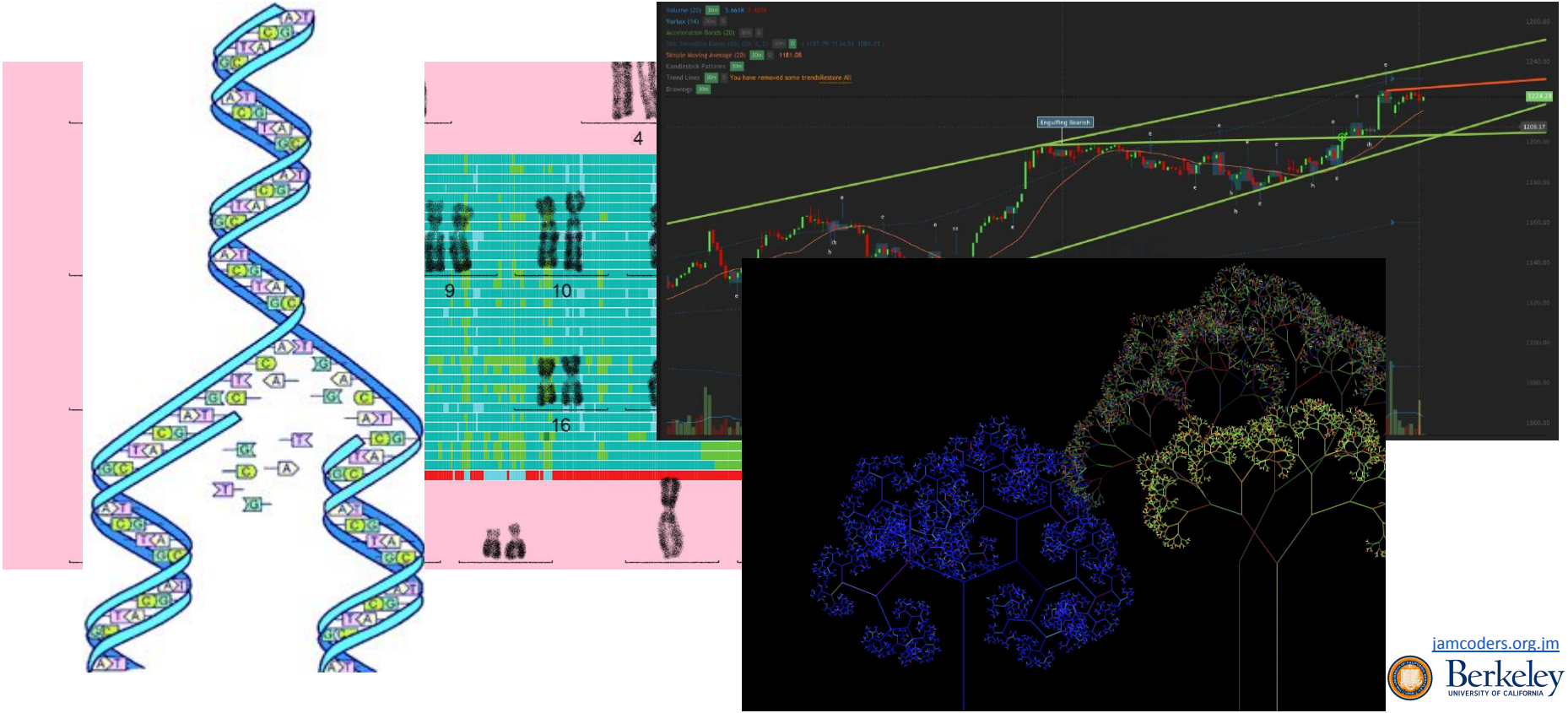
Her 11,500-year-old remains suggest that all Native Ame
trace their ancestry to the same founding population.

AlphaGo

Berkeley UNIVERSITY OF CALIFORNIA

Why Study Computer Science?

Not just by traditional Software Engineering...



Berkeley
UNIVERSITY OF CALIFORNIA

Why Study Computer Science?

Many reasons beyond just wanting to make websites and apps!

It’s also just fun :)

Programming?

What is programming?

"Programming" is designing and writing **code** to accomplish some task.

Smaller Tasks

- Compound Interest calculator
- Show an image on a screen
- Make a robot move its arm to the right

Bigger Tasks

- Browse videos on YouTube (build the YouTube app)
- Build a website for your class (like jamcoders.org.jm)
- Find the optimal route from Kingston airport to UWI

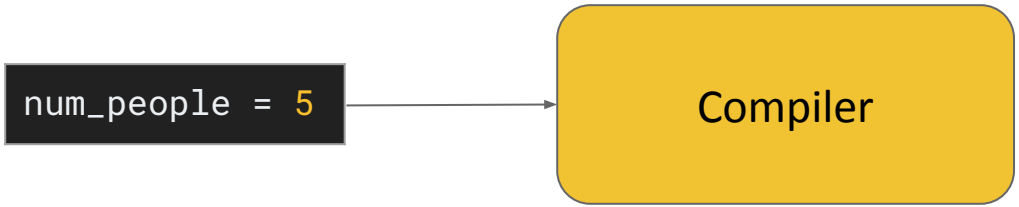
Running Code

What happens when you run code?

```
num_people = 5
```

Running Code

What happens when you run code?

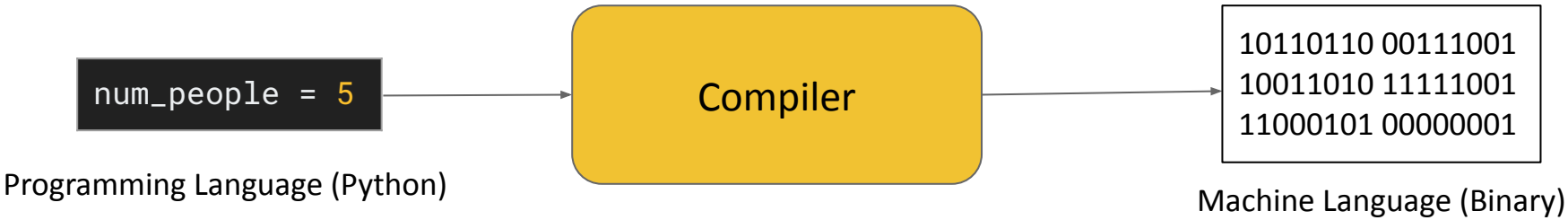


Running Code

What happens when you run code?

Running Code

What happens when you run code?



Running Code

What happens when you run code?

```
num_people = 5
```

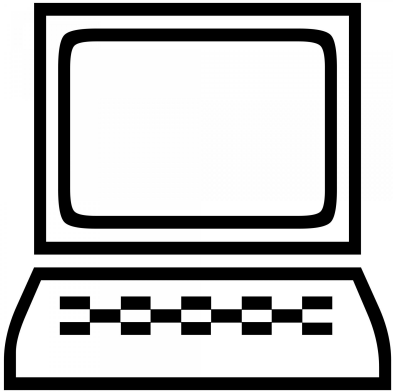
Programming Language (Python)

Compiler

10110110 00111001
10011010 11111001
11000101 00000001

Machine Language (Binary)

10110110 00111001
10011010 11111001
11000101 00000001



Running Code

What happens when you run code?

```
num_people = 5
```

Programming Language (Python)

10110110 00111001
10011010 11111001
11000101 00000001

Machine Language (Binary)

10110110 00111001
10011010 11111001
11000101 00000001

Output of the program:

- Calculate something
- Show an image
- Make a sound

etc.

Languages

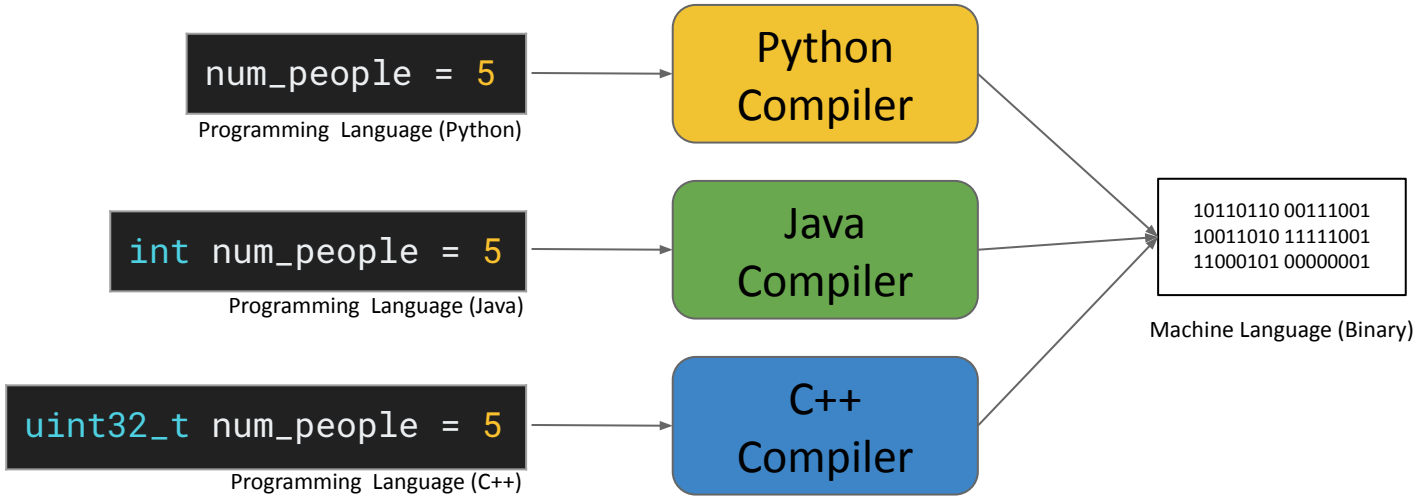
There are many different programming languages…

- Python
- Scratch
- Java
- C/C++
- Javascript
- Go
- R
- Swift

Languages

There are many different programming languages…

- Python
- Scratch
- Java
- C/C++
- Javascript
- Go
- R
- Swift



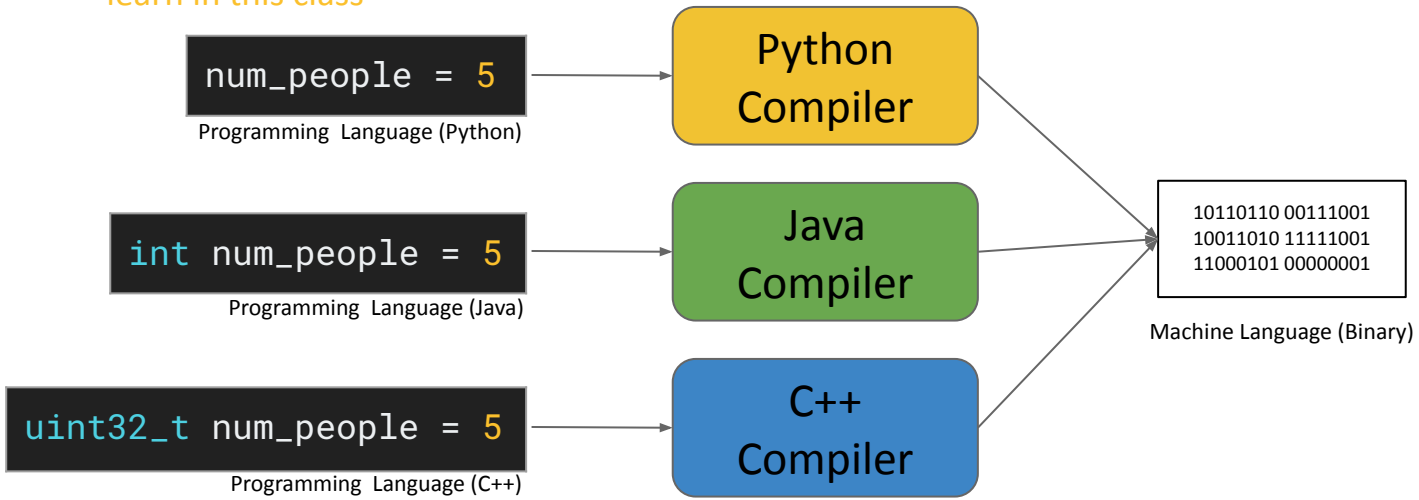
All of them get translated to the same “binary” language that the hardware understands.

Languages

There are many different programming languages…

- Python
- Scratch
- Java
- C/C++
- Javascript
- Go
- R
- Swift

Language we will learn in this class



All of them get translated to the same “binary” language that the hardware understands.

Why Learn Python?

Perfect language to start coding in

- Easy to read
- Very little “syntax”
  - **syntax**: the rules of grammar in a programming language
- Used extensively in the real world

YouTube

Walt Disney Pictures

Programming Environment

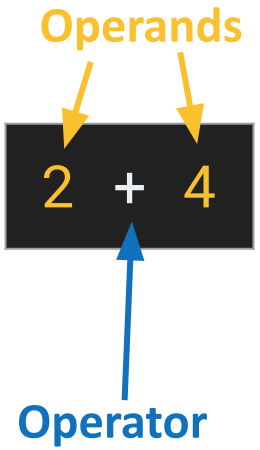
We will be coding in a “Notebook” or “Colab”…

Let’s jump to it

Expressions

Expressions

Expressions are combinations of *operators* and *values* **evaluating** to a single value.



Evaluates to 6.

Expressions

The **operands** in an expression can themselves be expressions. Expressions are always evaluated operand-first.

```
17 + (3 * 11)
```

```
17 + (33)
```

```
50
```

Python will simplify any expression, no matter how complicated, down until it gets to a single value!

Arithmetic Operators

The same operators that you're used to in math work in Python.

- Addition (+)

```
sum = 5 + 6
```

- Subtraction (-)

```
diff = 12 - 5
```

- Multiplication (*)

```
product = 8 * 4
```

- Division (/)

```
quot = 12 / 5
```

- Exponentiation (**)

```
power = 2 ** 6
```

(this is the same as 2^6)

Arithmetic Operators

If applying multiple operators, use parentheses to control the order they get evaluated, just like in math:

```
7 + ((20 / 4) ** 2)
```

7 + ((20 / 4) ** 2)

7 + (5.0 ** 2)

7 + 25.0

32.0

Handy New Arithmetic Operators

Floor division: // divides 2 numbers and rounds down to the nearest int.

x // y answers the questions “How many times does Y completely fit into X?”

Examples:

- 10 // 2 == 5
- 11 // 2 == 5
- 175 // 4 == 43

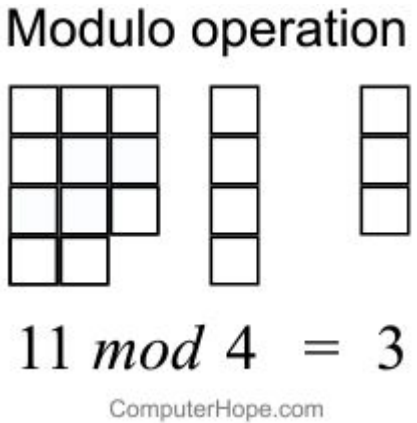
Handy New Arithmetic Operators

Modulo: % computes *remainder after division*

a % b

Examples:

- 11 % 5 == 1
- 120 % 100 == 20
- 1 % 5 == 1
- 24 % 2 == 0



Variables

Think back to your algebra class...

$$a = 5$$

$$b = 6$$

$$c = a + b$$

What's the value of c?

How did you know?

Python Variables

Python has the same concept.

A **variable** is a storage location filled with a value and associated with an identifier

```
num_apples = 5
```

Two New Variable Words...

1. “**Assign**” a variable
2. “**Reference**” a variable

Assign and Reference

Assigning a variable means storing a value in memory with a given name.

In variable assignment, the variable name is always on the left side of the equals sign:

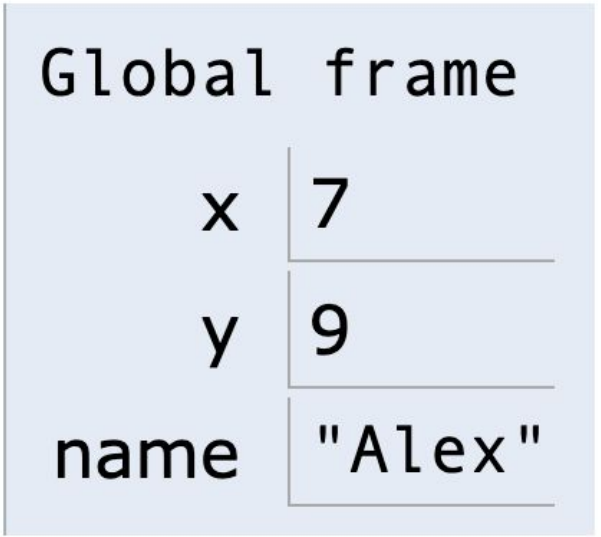
```
x = 7
y = x + 2
```

Assign and Reference

Assigning a variable means storing a value in memory with a given name.

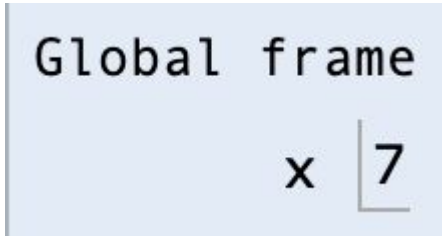
In variable assignment, the variable name is always on the left side of the equals sign:

```
x = 7
y = x + 2
name = "Alex"
```



“Reference”

Referencing a variable is how you read the value from memory.



Assign and Reference

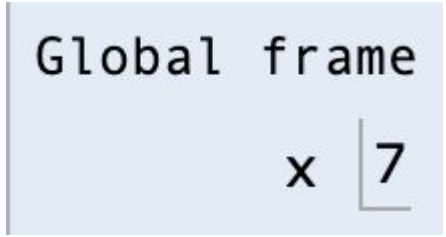
You can **reassign** a variable...give it a new value.

```
x = 7
print(x)
x = 9
print(x)
```

Assign and Reference

You can **reassign** a variable...give it a new value.

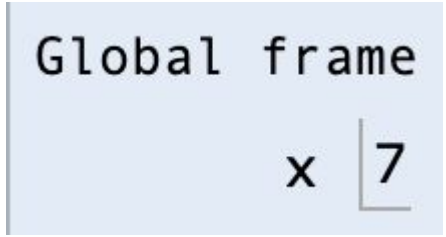
```
x = 7
print(x)
x = 9
print(x)
```



Assign and Reference

You can **reassign** a variable...give it a new value.

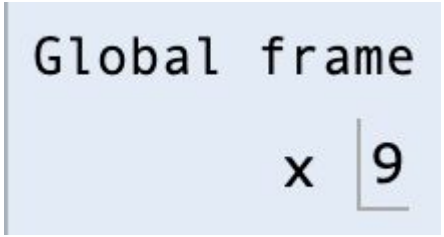
```
x = 7
print(x)
x = 9
print(x)
```



Assign and Reference

You can **reassign** a variable...give it a new value.

```
x = 7
print(x)
x = 9
print(x)
```



Assign and Reference

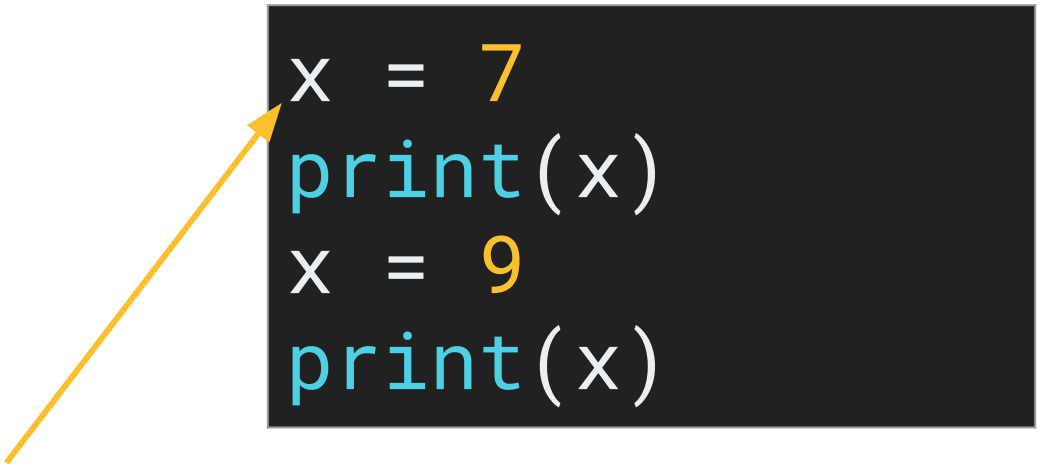
You can **reassign** a variable...give it a new value.

```
x = 7
print(x)
x = 9
print(x)
```

```
Global frame
        x | 9
```

Assign and Reference

Bonus word: “**declare**” (you won’t see this one often in this class)



The very first time you **assign** a variable a value is called “**declaring**” the value.

Variable Names in Python

1. Must be made up of letters, digits (0-9) or the underscore character _ .
2. May not start with a digit.
3. May not be a “reserved word”

Variable Names in Python

1. Must be made up of letters, digits (0-9) or the underscore character _ .
2. May not start with a digit.
3. May not be a “reserved word”

Variable Names in Python

1. Must be made up of letters, digits (0-9) or the underscore character _ .
2. May not start with a digit.
3. May not be a “reserved word”

How Python Runs

What Would Python Print? (WWPP)

Python runs and executes code *line-by-line*.

```
a = 5
b = 6
c = a + b
b = 10
print(c)
```

What Would Python Print? (WWPP)

Python runs and executes code *line-by-line*.

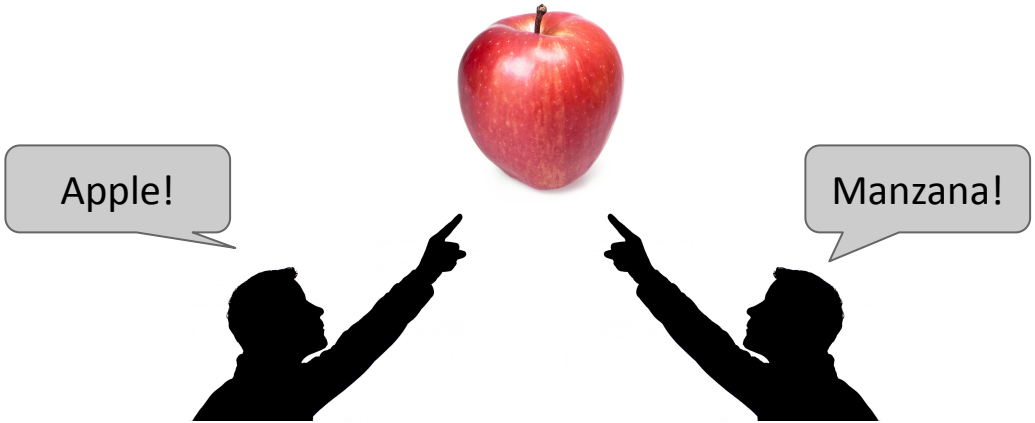
```
a = 5
b = 6
c = a + b
b = 10
print(c)
```

a	b	c
5		
5	6	
5	6	11
5	10	11
5	10	11

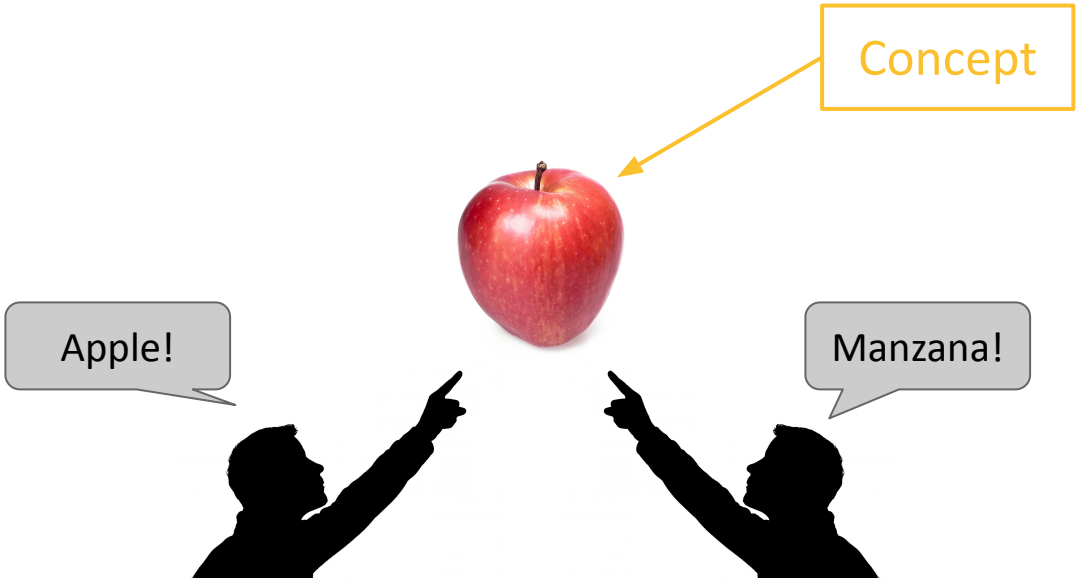
Extra Slides

A bit of learning theory...

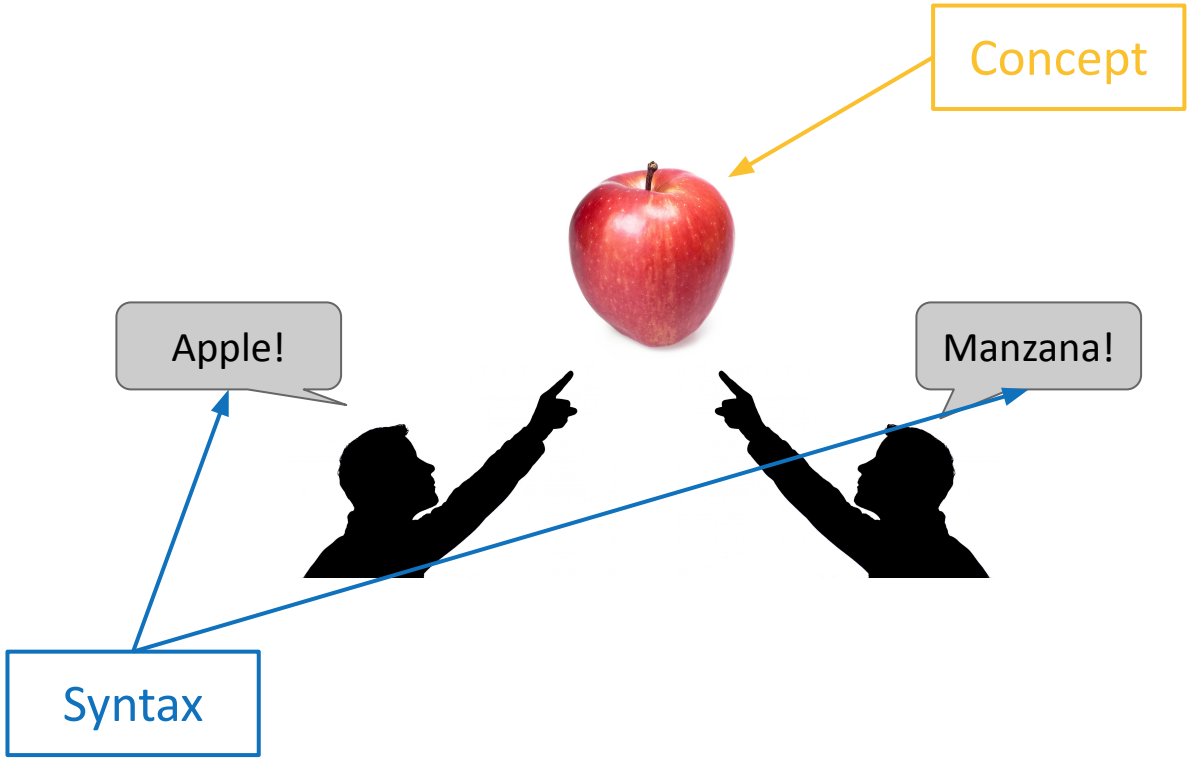
Concepts vs Syntax



Concepts vs Syntax



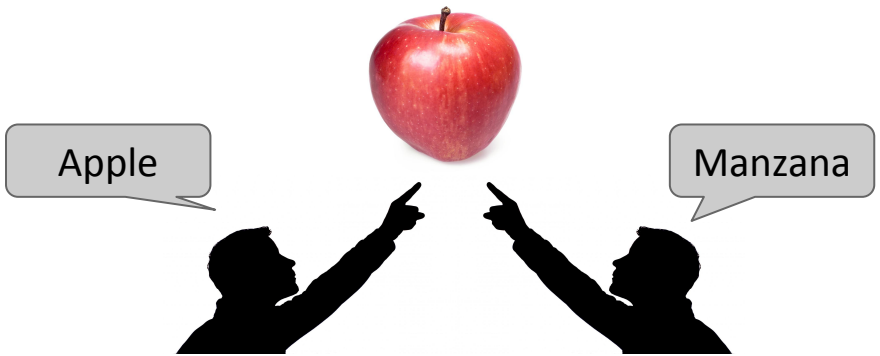
Concepts vs Syntax



Types of Learning in this Class

Type 1

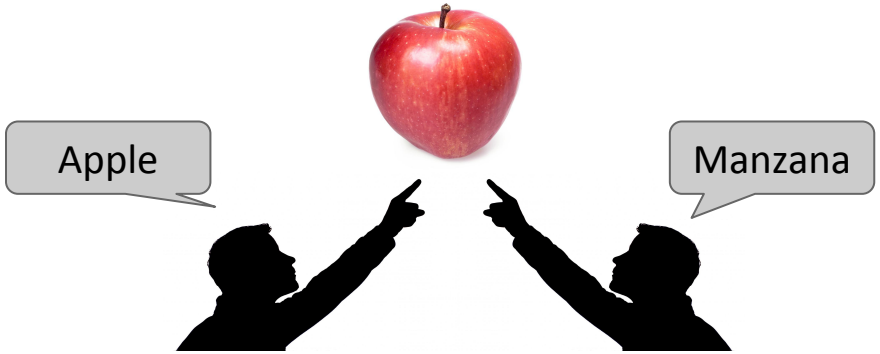
- Concept you already know
- New “syntax”



Types of Learning in this Class

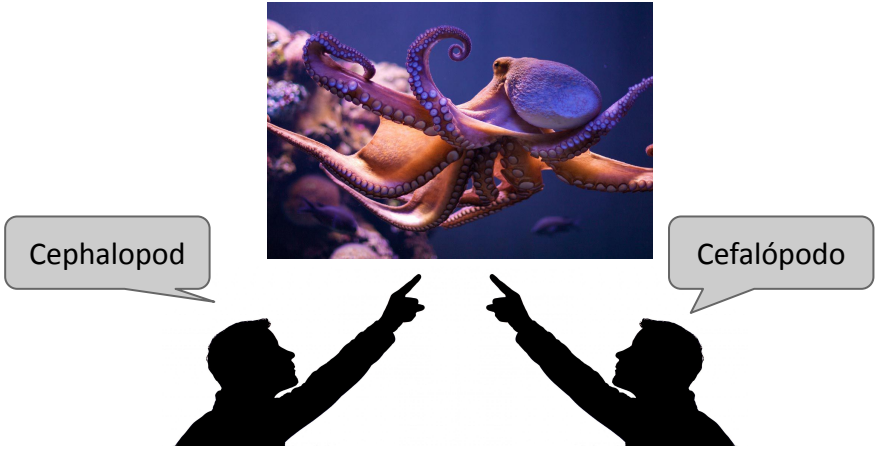
Type 1

- Concept you already know
- New “syntax”



Type 2

New Concept ***and*** New “syntax”



Types of Learning in this Class

Type 1

- Concept you already know
- New “syntax”

Examples

- Math operators (+, -, *, /)
- Variables (y = mx + b)
- Lists

Type 2

New Concept ***and*** New “syntax”

Types of Learning in this Class

Type 1

- Concept you already know
- New “syntax”

Examples

- Math operators (+, -, *, /)
- Variables (y = mx + b)
- Lists

Type 2

New Concept ***and*** New “syntax”

Examples

- Functions
- Recursion
- Control Flow

Types of Learning in this Class

Keep this difference in mind as you're learning.

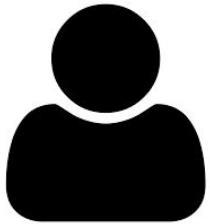
Make sure you understand the concept on its own, **and** how to apply it (the "syntax") in Python.

Software

Hardware

Software

Hardware



Understands Human instructions

```
while cond != True:
  instr = gen_inst()
  cond = eval(instr)
return instr
```

```
1100 1001
1101 0111
0010 1101
1001 1001
```

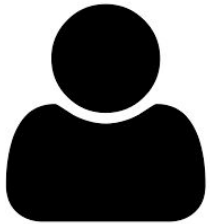
Understands "Binary Code" Instructions

Compiler

Software

Hardware

This Class



Understands Human instructions

```
while cond != True:
  instr = gen_inst()
  cond = eval(instr)
return instr
```

```
1100 1001
1101 0111
0010 1101
1001 1001
```

Understands "Binary Code" Instructions

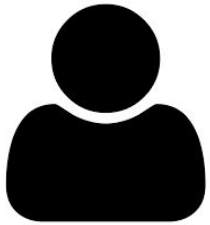
Compiler

Software

Hardware

This Class

Python



```
while cond != True:
  instr = gen_inst()
  cond = eval(instr)
return instr
```

Understands Human instructions

1100 1001
1101 0111
0010 1101
1001 1001

Understands "Binary Code" Instructions

Compiler