

01-18-2023

# CS051A

## INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

### 1: Introduction

---



Alexandra Papoutsaki

she/her/hers

Lectures



Zilong Ye

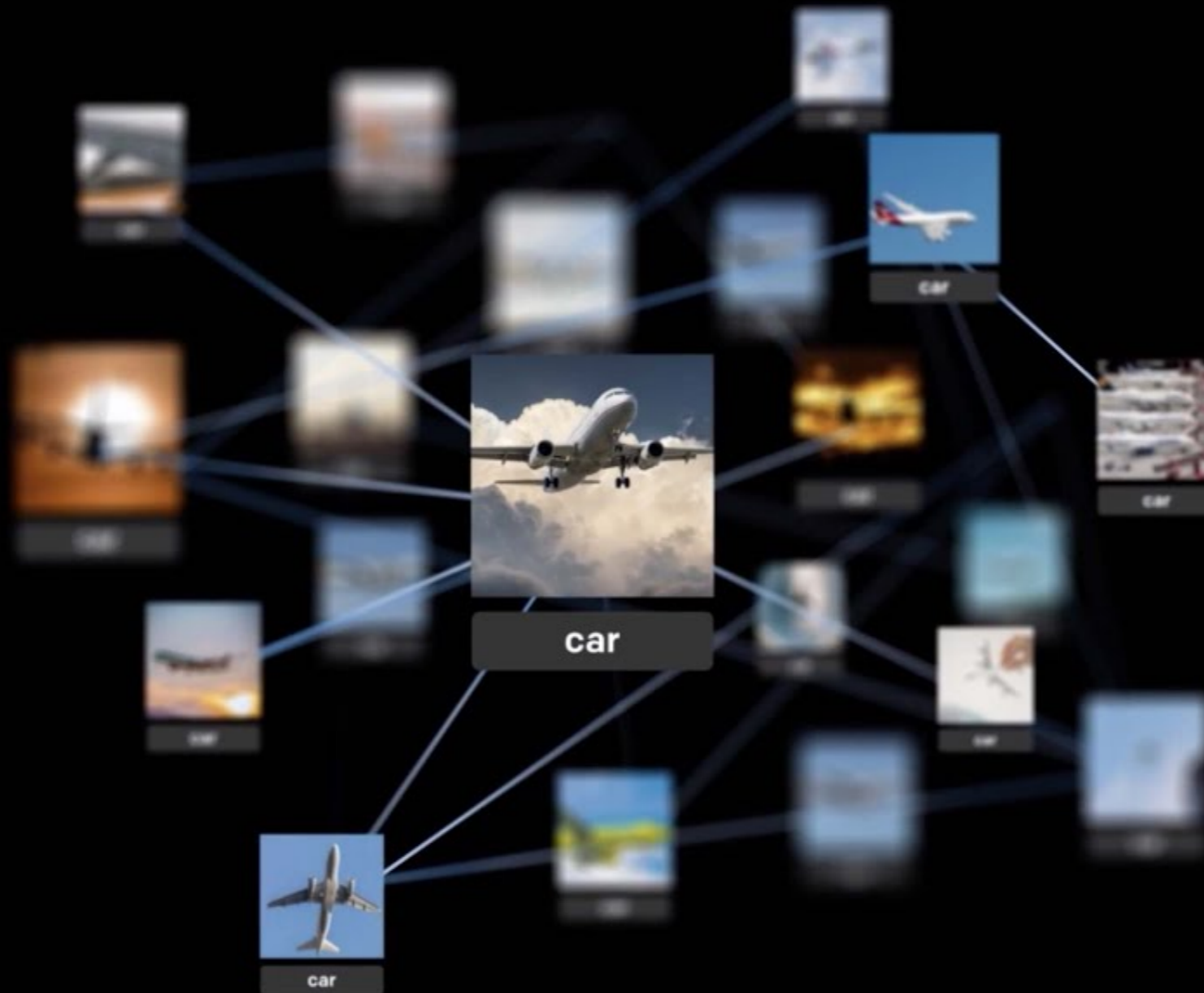
he/him/his

Labs

## Lecture 1: Introduction

- ▶ Introduction
- ▶ Logistics
- ▶ Intro to Python

# DALL-E-2




# ChatGPT


+ New chat

New chat


ChatGPT for CS students


ChatGPT: Language m  


*Due to high demand on our systems, previous conversations are temporarily unavailable*

 Clear conversations

 Dark mode

 OpenAI Discord

 Updates & FAQ

 Log out

AL what is chatgpt



ChatGPT is a variant of the GPT (Generative Pre-trained Transformer) language model developed by OpenAI. It is a neural network-based model that is trained on a large corpus of text data to generate human-like text. ChatGPT is specifically designed to generate text in the context of conversational interfaces, such as chatbots, and it can be fine-tuned on specific tasks, such as answering questions or responding to prompts, by training it on a smaller dataset specific to that task.



AL what corpus was used to train it?



ChatGPT was trained on a diverse corpus of text data, which includes a wide range of internet text such as articles, books, websites and conversational transcripts. This corpus contains a huge amount of text data, that is used to train the model to generate natural and coherent text that resembles human writing. The goal of training such a model with a diverse corpus is to make the model able to adapt and generalize to different types of language and to generate more accurate and diverse outputs.

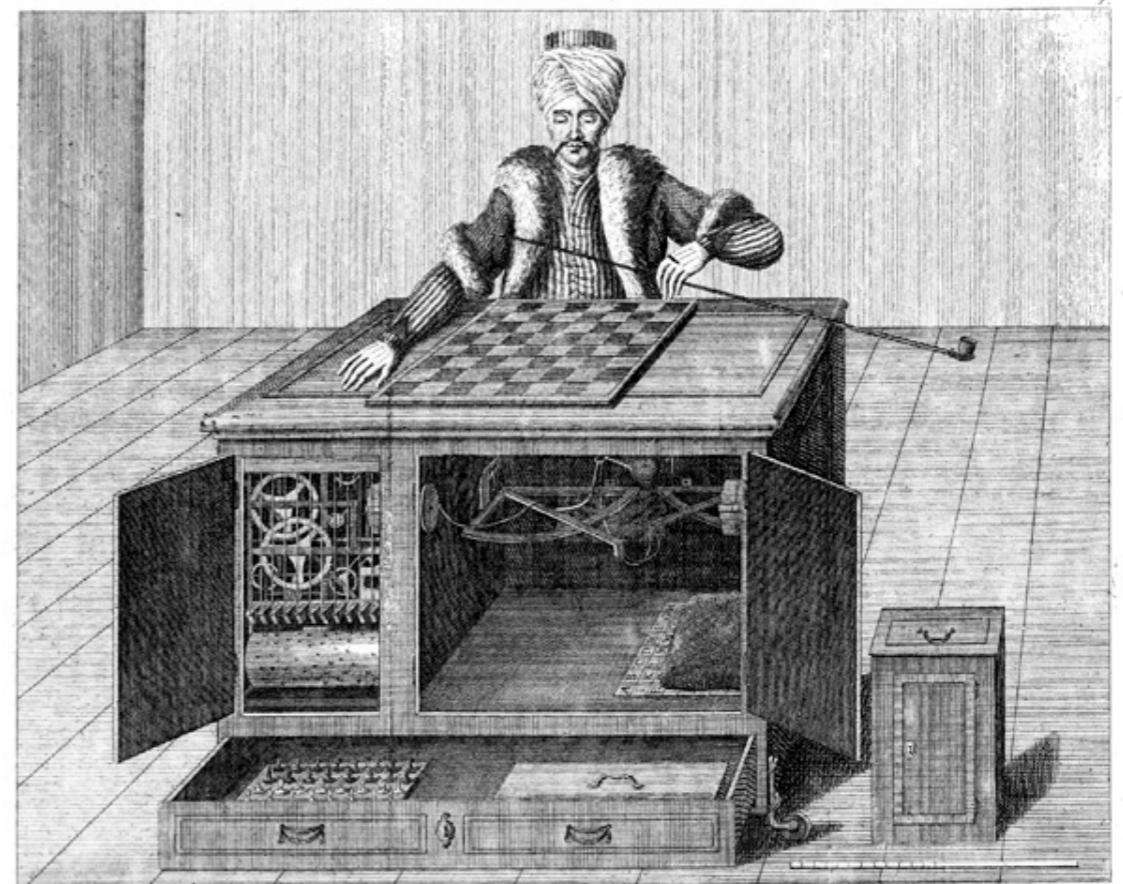




# Humans have long been fascinated with intelligent machines



Antikythera mechanism, ~150BC

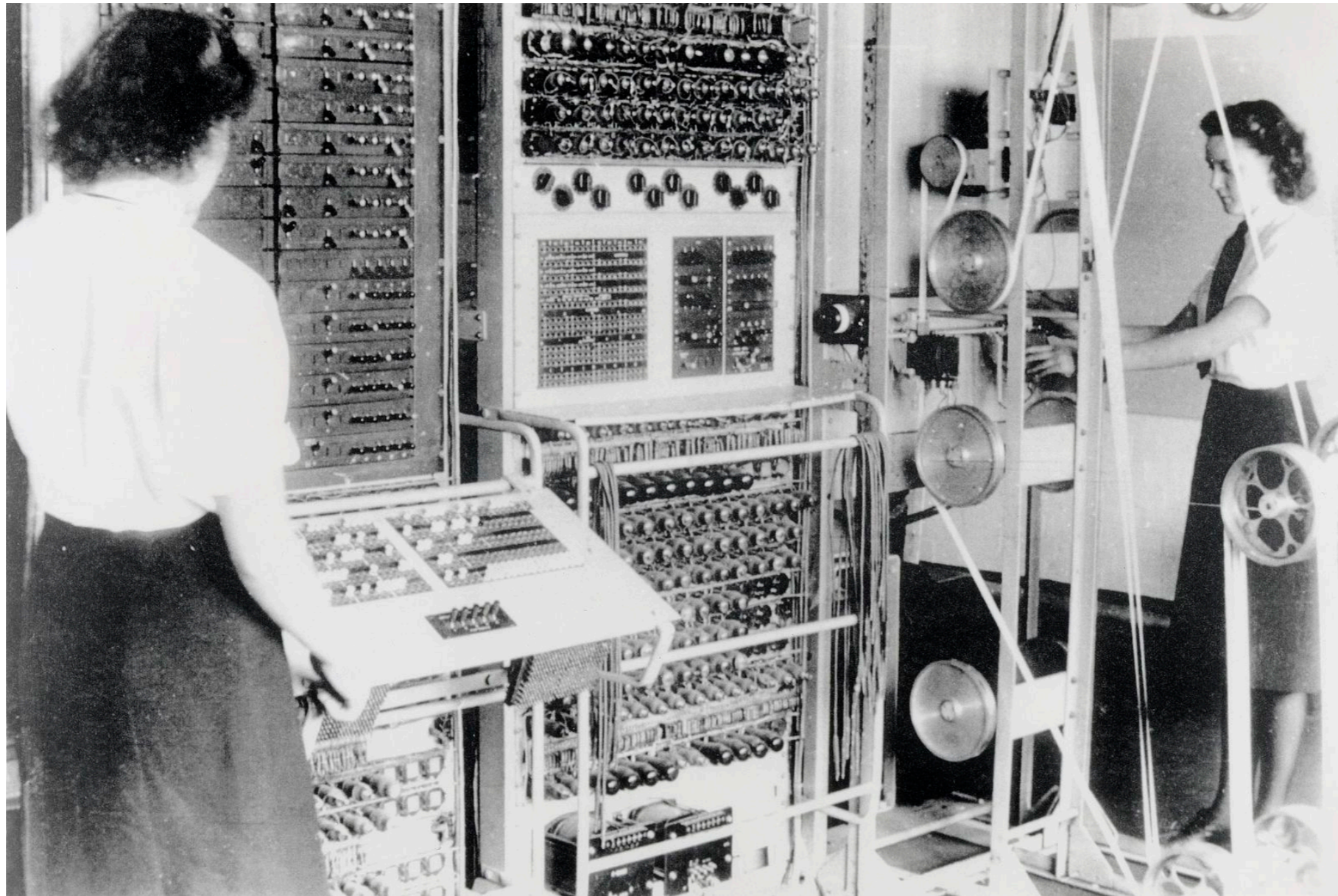


*W. de Kempelen del. Che a Mechel, exaud. Basilea. P.G. Piatz, fec.*  
*Der Schachspieler, wie er vor dem Spiel gezeigt wird von vorn. Le Joueur d'echecs, tel qu'on le montre avant le jeu, par devant.*

Chess Player or "Turk", 18th century



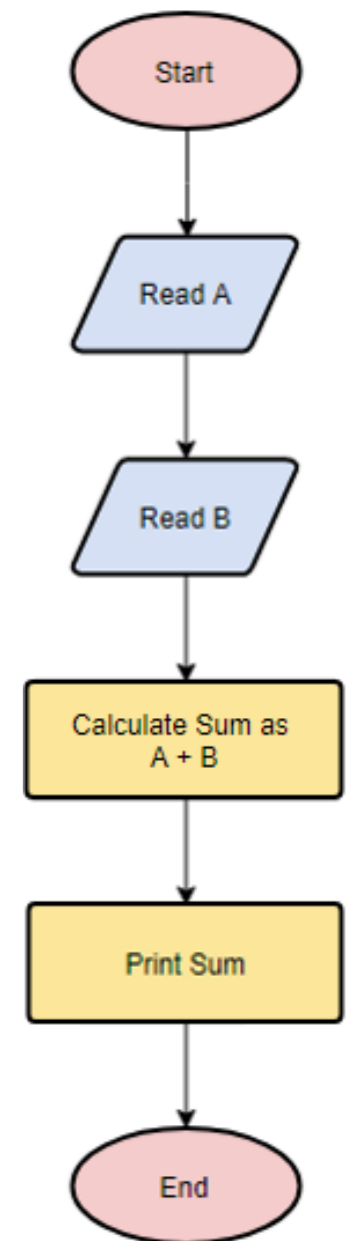
# Computers can be programmed to solve problems



Colossus, the first electronic digital programmable computing device, was used to break German ciphers during World War II.

1st learning goal: design algorithms for different problems

- ▶ **Algorithm:** A step-by-step list of instructions that if followed exactly will solve the problem under consideration.
  - ▶ In that sense, a cooking recipe is an algorithm.
- ▶ Algorithms can be expressed in different notations: natural languages, pseudocode, flowcharts, programming languages, etc.



## 2nd learning goal: program in Python

- ▶ **Programming**: the process of taking an algorithm and encoding it into a programming language so that it can be executed by a computer.
- ▶ There are tons of programming languages, e.g., Python, Java, JavaScript, C, C++, etc.
  - ▶ In this course, we will learn Python.
  - ▶ We don't assume any prior computer science, programming or science background.





3rd learning goal: explore topics in Artificial Intelligence

- ▶ **Artificial Intelligence:** intelligence demonstrated by machines, as opposed to natural intelligence displayed by animals and humans.
- ▶ Numerous applications you are already familiar with: recommendation systems, virtual assistants, self-driving cars, etc.
- ▶ Not only learn about fascinating applications but also consider ethical implications.



---

Is CS51A representative of what being a CS major will be?

- ▶ Yes and No. It's a big world out there.
- ▶ Some computer scientists might not even work with computers or program at all.
- ▶ Practical applications range from low-level hardware to software.
- ▶ Interdisciplinary work with psychologists, policy makers, etc.
- ▶ CS51A builds fundamental understanding of computational thinking, programming, AI, and ethical implications of tech.

## Lecture 1: Introduction

- ▶ Introduction
- ▶ **Logistics**
- ▶ Intro to Python



### A typical week

- ▶ Monday and Wednesday lectures.
- ▶ Monday or Tuesday evening labs.
- ▶ Weekly assignments due on Sunday.
  - ▶ Most will be programming-based in Python.
  - ▶ Readings on ethical-related topics.

### Course website

- ▶ <https://cs.pomona.edu/classes/cs51a/>
  - ▶ Make sure to bookmark it.
- ▶ Contains all necessary information about the course, schedule, links to lectures, notes, and code, grading, etc.
- ▶ Read the [syllabus](#) carefully!
- ▶ Consult it on a weekly basis.

### Textbook

- ▶ How to Think Like a Computer Scientist: Interactive Edition. Brad Miller and David Ranum, based on original work by Jeffrey Elkner, Allen B. Downey, and Chris Meyers.
- ▶ It is available [online for free](#).

## Our mentors



Elshiekh Ahmed  
*he/him/his*  
BBICS mentor



Catherine Byen  
*she/her/hers*



Alej Castañeda  
*she/her/hers*



Caleb Kim  
*he/him/his*



Hasana Parker  
*she/her/hers*  
BBICS mentor



Sarah Shader  
*she/her/hers*



Chau Vu  
*she/her/hers*



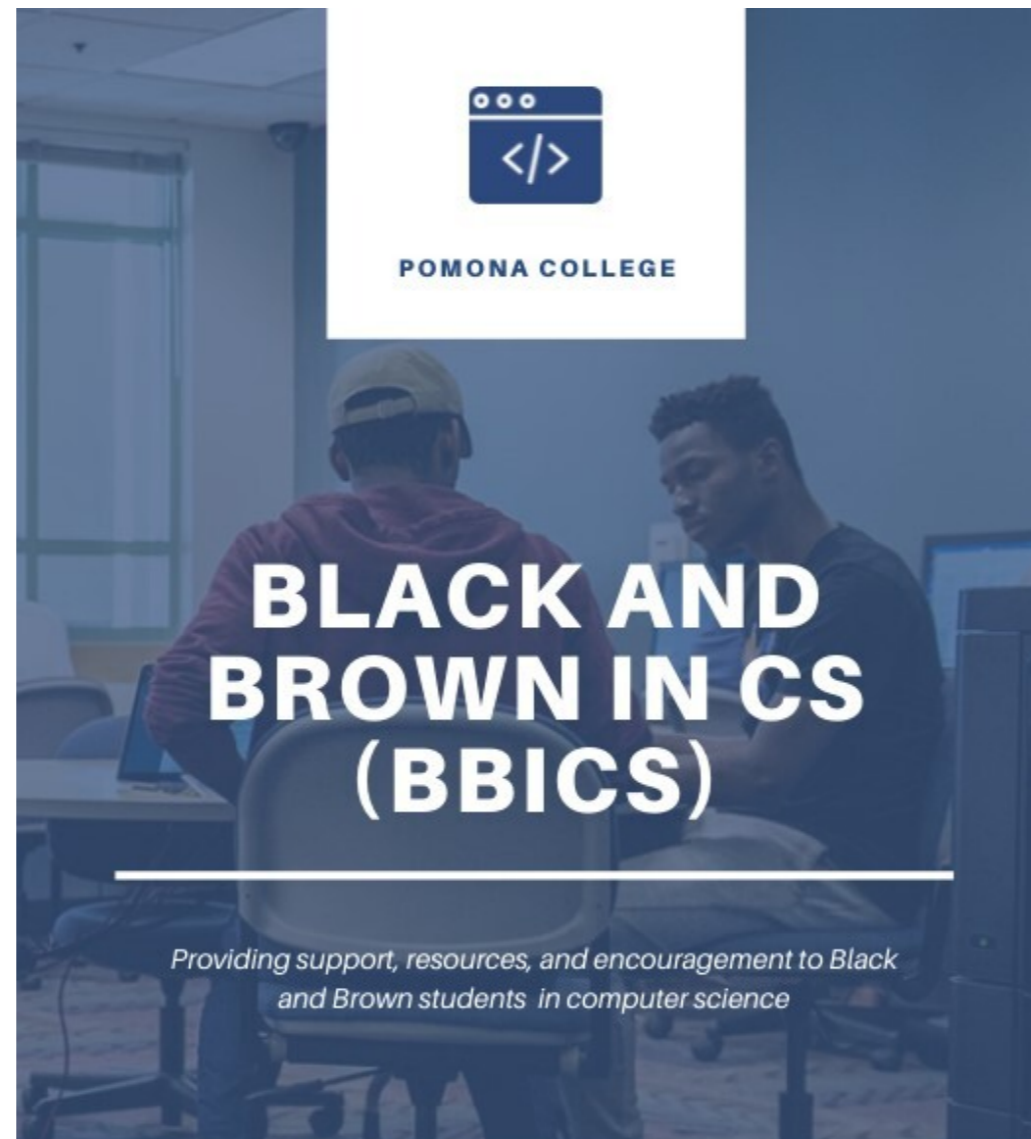
Rachel Yang  
*she/her/hers*

### Slack Channels

- ▶ If registered, already invited to cs51a-spring2023 channel.
  - ▶ You need a Pomona Slack account.
  - ▶ Please let me know if you have not been added yet.
  - ▶ You can post questions anonymously, too.
- ▶ Department-wide slack workspace:  
<https://tinyurl.com/PomonaCSSlack>

### CS student liaisons

- ▶ Gabriel Konar-Steenberg '23
- ▶ Hannah Mandell '23
- ▶ Aldo Ruiz Parra '25
- ▶ Maymuunah Quasim '24
- ▶ Rachel Yang '24



## What we do?

- > Personal and academic mentoring
- > Practice Interviews
- > Monthly newsletter w/ shared resources
- > Monthly speakers

contact | [bbicspom@gmail.com](mailto:bbicspom@gmail.com)



### P-ai

- ▶ Project-based AI student organization
- ▶ <https://www.p-ai.org/>
- ▶ Planned events for the semester:
  - ▶ AI Art workshop
  - ▶ Resume workshop
  - ▶ Celebration of Pi day
  - ▶ AI for creative writing

Sakai survey is due this Friday at midnight

- ▶ Getting to know you
- ▶ Respond to questions you feel comfortable. Instructors have provided their responses to the questions in return.

### Grade calculation

- ▶ 40%: Assignments and lab sessions
  - ▶ No late work or else cascading overlapping deadlines (stressful for all parties involved)
- ▶ 35%: Two midterms
- ▶ 20%: Final exam
- ▶ 5%: participation
  - ▶ Expected to attend lectures and labs and ask questions.
  - ▶ Includes a mid-semester group presentation on a topic of choice on ethics + AI.

### Honor code

- ▶ Work by yourself on a problem before seeking help.
- ▶ Avoid Internet help beyond the course website, Python's official documentation, and textbook.
- ▶ Help among classmates is ok, but...
  - ▶ A peer should never touch your computer or look at your screen.
  - ▶ Don't take away someone's "a-ha!" moment.
- ▶ TA expectations
  - ▶ TAs are there to help you reach your own answers.
  - ▶ Don't ask them to work outside of mentor sessions.
  - ▶ Use slack outside office hours/mentor sessions.

### How to succeed in this course

- ▶ Come to class and lab and ask questions.
- ▶ Write down confusing things or questions but not every detail (this is why we have slides and notes).
- ▶ If you are confused or struggling within or outside class, let the instructors or a TA know. We are here to help.

## Lecture 1: Introduction

- ▶ Introduction
- ▶ Logistics
- ▶ Intro to Python

### Programming languages

- ▶ **High-level** vs **low-level** (machine or assembly) languages.
- ▶ Python, as well as Java and C++, are high-level.
- ▶ Machine languages encode instructions in binary (0s and 1s). Computers can only execute programs written in machine language.
- ▶ High-level languages are slower but much more **readable** and **portable**.



# Low-level language vs high-level languages

```
lcfi2:
    movl    %edi, -4(%rbp)
    cmpl   $0, -4(%rbp)
    jle    LBB0_2
## BB#1:
    leaq   L_.str(%rip), %rdi
    movb  $0, %al
    callq _printf
LBB0_2:
    xorl   %eax, %eax
    retq
L_.str:
    .asciz "x is a positive number"
```

```
if (x>0):
    print ("x is a positive number")
```

## Interpreters vs compilers



Figure 1.1: An interpreter processes the program a little at a time, alternately reading lines and performing computations.

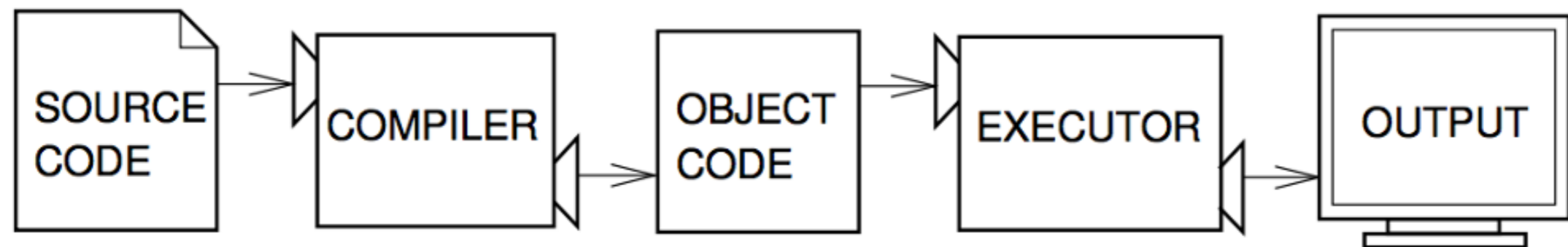


Figure 1.2: A compiler translates source code into object code, which is run by a hardware executor.

---

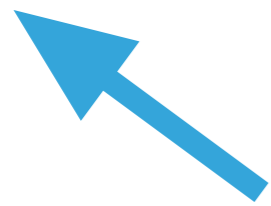
### Shell mode

- ▶ You type Python expressions into the Python shell, hit enter/return key, and the interpreter immediately shows result (if there is one).
- ▶ Great for testing small code similar to scratch paper

```
>>> 2+3
```

```
5
```

```
>>> █
```



Python prompt

# Python makes for a great calculator

- ▶ Just by opening the Python shell, we can do all sorts of math:
  - ▶ Addition: +
  - ▶ Subtraction: -
  - ▶ Multiplication: \*
  - ▶ Division: /
  - ▶ Power or exponentiation: \*\*
  - ▶ Mod or remainder: %

## Basic math calculations

```
>>> 4+4
```

```
8
```

```
>>> 10-20
```

```
-10
```

```
>>> 15*20
```

```
300
```

```
>>> 20/4
```

```
5.0
```

```
>>> 10+4*2
```

```
18
```

```
>>> (10+4)*2
```

```
28
```

```
>>> 2**10
```

```
1024
```

```
>>> 2**30
```

```
1073741824
```

# Operator precedence

- ▶ Python follows the normal operator precedence you're used to for math:
  - ▶ things in parentheses get evaluated first,
  - ▶ `**` is next,
  - ▶ `%`, `*`, and `/` next,
  - ▶ `+` and `-` last.

Why are these different?

```
>>> 4+4
```

```
8
```

```
>>> 20/4
```

```
5.0
```

- ▶ “True division” always results into a floating-point number or float.



### Numeric types

- ▶ `int`: integer numbers, e.g., `-15`, `0`, `47`
- ▶ `float`: floating-point numbers, e.g., `-15.0`, `0.3`, `46.999`
- ▶ Every value has an associated type.

# Statements and Expressions

- ▶ **Statement:** an instruction that Python can execute
  - ▶ A program is just a sequence of statements separated by new lines.
- ▶ **Expression:** a combination of values (literals) and operators. Expressions need to be evaluated by the interpreter into a value.
  - ▶ Incomplete definition.
  - ▶ Everything we have seen so far has been an expression, e.g., the expression `3+5` evaluates to the value `8`.
- ▶ Python is a "strongly typed" language: every expression in Python has a type. We have seen two so far, `int` and `float`.
- ▶ If any number within an expression is a `float`, the whole expression will be a `float`.

### It's BBQ time!



- ▶ You are having a party and you're trying to figure out how many hot dogs to buy. Here are some facts:
  - ▶ Angie isn't a big fan of hot dogs, so she'll only eat 1.
  - ▶ Jasmine generally eats 2.
  - ▶ Chris always eats twice as many as Jasmine.
  - ▶ Brenda eats one less than Chris.
  - ▶ Wenting eats half as many as Brenda at the party and also likes to take one extra for home.
- ▶ Try to do this on paper: 13 ( $=1+2+4+3+3$ , assuming that if someone eats half a hot dog, we still have to count the whole thing).

# Variables

- ▶ **Variables:** containers we use to store values.
- ▶ A variable is essentially a storage for a value.  

```
>>> angle = 1  
>>> angle
```

1
- ▶ **Assignment** statements link a variable name or identifier (left-hand) to value (right-hand). It tells the interpreter to do something, but does NOT represent a value.
- ▶ 

```
>>> angle = 2  
>>> angle
```

2
- ▶ Expression is a combination of values (literals), variables (identifiers), and operators
  - ▶ Still incomplete definition

### Hot dog calculations using Python shell

```
>>> angie = 1
```

```
>>> jasmine = 2
```

```
>>> chris = 2 * jasmine
```

```
>>> brenda = chris - 1
```

```
>>> wenting = brenda/2 +1
```

```
>>> total_hotdogs = angie + jasmine + chris + brenda +  
wenting
```

```
>>> total_hotdogs  
12.5
```

# Integer division

- ▶ Why 12.5? Remember that division is 'real' in Python.
- ▶ `brenda/2 = 1.5`
- ▶ We can fix this with **integer division**, `x // y`, which truncates (ie. decimal places are dropped) the result.

```
>>> 11//2  
5
```

```
>>> 10//3  
3
```

```
>>> 11//3  
3
```

```
>>> 11.0//2  
5.0
```

# How does integer division help us?

```
>>> wenting = (brenda + 1) // 2 + 1
```

- We add one to force it to round up
- If it's an odd number, it does what we want:

```
>>> 3 // 2
```

```
1
```

```
>>> (3 + 1) // 2
```

```
2
```

- if it's an even number, it doesn't change the answer:

```
>>> 4 // 2
```

```
2
```

```
>>> (4 + 1) // 2
```

```
2
```

### Putting everything together

```
>>> angie = 1
```

```
>>> jasmine = 2
```

```
>>> chris = 2 * jasmine
```

```
>>> brenda = chris - 1
```

```
>>> wenting = (brenda+1) // 2 + 1
```

```
>>> total_hotdogs = angie + jasmine + chris + brenda +  
wenting
```

```
>>> total_hotdogs  
13
```



### Naming variables

- ▶ Generally, you want to give good names (identifiers) to variables.
  - ▶ `x` and `y` are not good names unless they represent `x` and `y` coordinates :)
- ▶ Variable names should be all lowercase.
- ▶ Multiple words should be separated by an `'_'` (underscore).
  - ▶ e.g., `total_hotdogs`

### Change of plans

- ▶ Let's assume Jasmine skipped breakfast and now she wants to have 4 hot dogs.
- ▶ We would have to re-enter all lines (except first one) :(
- ▶ We already had to do this once to change for the // and it was annoying. We don't want to have to keep doing it!

## Program mode

- ▶ Write source code in a .py and run it.



```
bbq.py x
1  # This program figures out the number of hot dogs
2  # needed for a BBQ
3  angie = 1
4  jasmine = 2
5  chris = 2 * jasmine2
6  brenda = chris - 1
7  wenting = (brenda+1)//2 + 1 # add 1 to brenda to round up
8
9  total_hotdogs = angie + jasmine + chris + brenda + wenting
```

- ▶ No line-by-line feedback, we would need to print variables to see contents.

### Run program in Python console (shell)

- ▶ If you want to run your source code file AND have access to the variables so that you continue interacting with them, right click anywhere on your file, select "Run file in Python console".
- ▶ Now you have access to the variables in the Python console (shell)

### Making our programs more readable

- ▶ Use whitespaces and blank lines to make code more readable.
- ▶ Use comments (start with #) to leave notes to yourself and other programmers.
- ▶ Python will ignore everything from # to the end of the line.
- ▶ You can put comments on lines by themselves or have in-line short comments at the end of a line of code.
- ▶ Comments are *extremely* important. You will be required to put them in your programs for this course.

### PyCharm IDE

- ▶ **IDE**: Integrated **D**evelopment **E**nvironment.
- ▶ Text editor to edit and save source code files.
- ▶ Tools for running, debugging, and navigating code in "projects"
- ▶ For now, mostly using the "Python Console" and text editor
- ▶ Setup instructions available on the website for reference and in first lab.
- ▶ You can customize the editor and rearrange it how you like. In fact, please do make it yours as much as possible.

## Resources

- ▶ [DALL·E 2 Explained](#)
- ▶ <https://chat.openai.com/>
- ▶ Textbook: Chapter [1](#) and [2](#)
- ▶ [bbq.txt](#)

## Homework

- ▶ You don't have to start assignment 1 until you do lab 1 next Monday or Tuesday.
- ▶ BUT, only for this week, you can start the Ethics reading (included in the assignment 1 writeup) early.
- ▶ In subsequent weeks, we will be working on code and ethics reading simultaneously.