# CS051A

## INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

## 1: Introduction

**David Kauchak**
he/him/his
Lectures

**Alexandra Papoutsaki**
she/her/hers
Lectures

**Zilong Ye**
he/him/his
Labs

Happy New Year, welcome to a new semester and to your first CS course! I know that being online for the first two weeks is not what we wished for, especially after being in person last semester, but I hope that our community will confirm that it's safe and we'll go back to a sense of normalcy soon. Hopefully, you are all here for CS51A-Intro to Computer Science with Topics in AI and you are already enrolled in the class. My name is Alexandra Papoutsaki and most students call me Professor Papoutsaki. My pronouns are she/her/hers. I am one of the instructors of this course along with Dr. Dave and Professor Ye. Dr. Dave stopped by today to say hi. Dr. Dave and I will be leading the lectures, with me being mostly in the first half of the semester. Professor Ye will be leading the Monday and Tuesday evening labs. To register for this class, you need to also register for either the Monday and Tuesday lab for which attendance is mandatory.

Lecture 1: Introduction

▸ **Introduction**

▸ Logistics

▸ Intro to Python

I want to start by giving you a sense of what this class is about before we move into talking about logistics. At any point, you can use the chat or unmute yourself to ask me questions.

## Two AIs talk about why the universe exists



Let's start by watching a video about two AIs, that is artificial intelligences, talk about why the universe exists. [PAUSE] What did you think? What powers the dialogue behind this video is GPT-3, a machine learning model which you give it a small amount of text and it auto-generates relevant text thus creating this dialogue. The second relevant technology is synthesia.io a video platform you provide text to and it creates animated avatars that read this text aloud. These technologies are at the heart of what we will learn in this course: fundamentals of computer science with applications in artificial intelligence. Let me try to give you a short motivation for the study of these topics.

# Humans have long been fascinated with intelligent machines



Antikythera mechanism, ~150BC
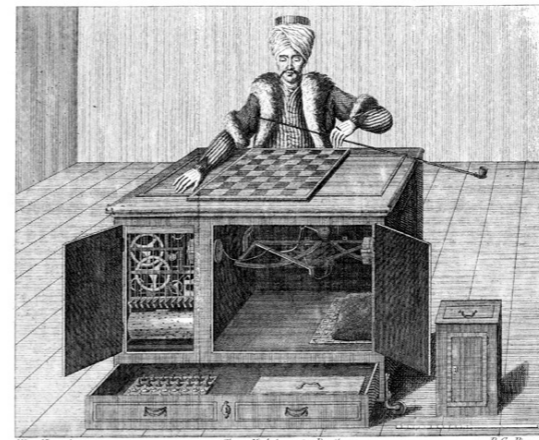


Chess Player or "Turk", 18th century

Humans have long been fascinated by the idea of using intelligent machines that would make mathematical and astronomical calculations, automate labor, entertain humans, and even pass as humans themselves. Archaeologists and historians have documented such devices since ancient China, Babylonia, Egypt, and Greece.

## Computers can be programmed to solve problems
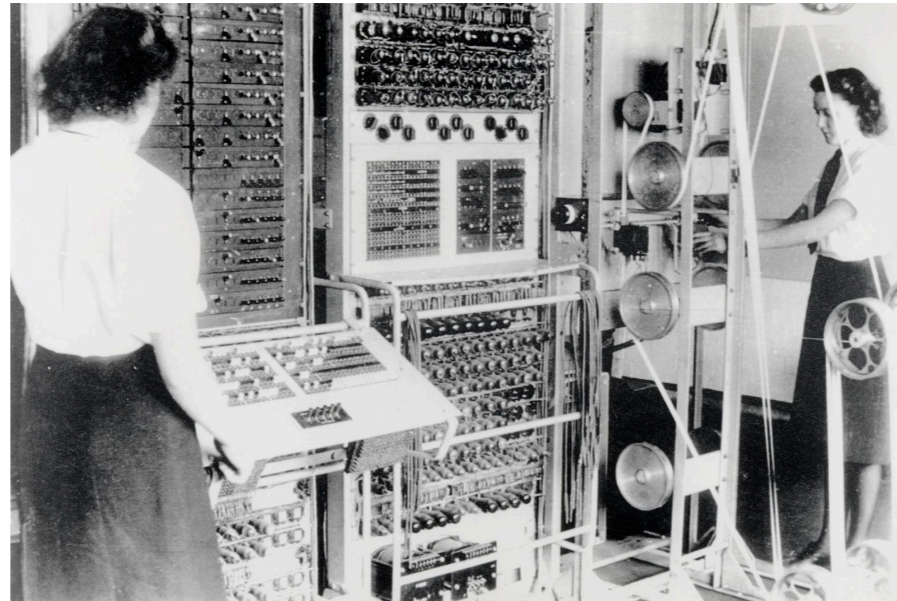


Colossus, the first electronic digital programmable computing device, was used to break German ciphers during World War II.

By the end of the first half of the 20th century, mechanical computers were substituted by a new generation of electronic computers that revolutionized scientific calculations and warfare. Around that time, there was also a shift around the way we talk about computers to refer to machines instead of people, usually women, that carry computations. Today, we think of computers as machines that can be *programmed* to carry out sequences of mathematical and logical operations automatically in order to *solve problems*. The problems that computers can solve vary greatly in nature and application and they touch virtually every sector of modern life: education, healthcare, transportation, communication, are just a few examples of sectors that have been transformed by computers.

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.

Start → Read A → Read B → Calculate Sum as A + B → Print Sum → End
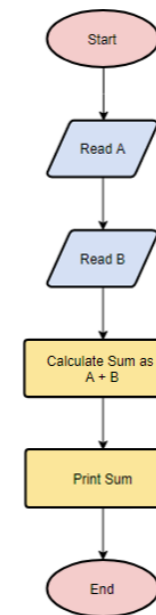
The first learning goal of this course will be to learn how to design algorithms for different problems. An algorithm is a step-by-step list of instructions that, if followed exactly, will solve the problem under consideration. If you think about it, you already encounter algorithms all the time. For example, if you want to make spaghetti, there's a certain set of steps you need to follow in a particular order. First, you'll need to boil a pot of water. Once it's boiling, you then add the spaghetti and cook it for a set amount of time, stirring occasionally. Once it's finished, you drain the water, then it's ready to be served with a sauce of your choice. Because you followed these steps in a particular order, you reached your desired outcome: a pasta dish. Algorithms can be expressed in many kinds of notation, including natural languages, pseudocode, flowcharts, programming languages etc. For example, in this picture, you can see a flowchart that visualizes an extremely simple algorithm that calculates the sum of two numbers. The algorithm reads two numbers, adds them up, and prints their sum.

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.

The second learning goal is to teach you how to program, that is learn how to take an algorithm and encode it into a programming language so that it can be executed by a computer. There are hundreds of programming languages out there. You might have heard some of them: Python, Java, JavaScript, C, C++, etc. In this course, we will learn Python and use it to make our computer understand the solution we designed for different types of problems. Our course doesn'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.

Going back to the fascination of humans with intelligent machines, the 1950s saw the birth of a new field, that of artificial intelligence. AI refers to intelligence demonstrated by machines, as opposed to natural intelligence displayed by animals and humans. You are already familiar with popular AI applications: recommendation systems when you watch Netflix or YouTube videos, virtual assistants such as Siri and Alexa, self-driving cars, etc. The third learning goal of this course will be to explore fundamental ideas in artificial intelligence, while reinforcing our algorithmic and programming skills. At the same time, we want to touch on how technology in general and artificial intelligence specifically can have severe ethical ramifications and can help or intensify structural inequalities in our society.

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.

So with these three learning goals in mind, I want to also touch on whether CS51A is representative of what being a CS major will be like. And the answer is yes and no. CS51A is by no means a survey course in cs. Computer science is a new and rapidly growing field that tackles a number of problems: largely speaking, it's interested in the study of computation, automation, and information. Some computer scientists are doing theoretical work and ask questions such as what problems are even computable? They often don't use computers much and work a lot with pen and paper or on the whiteboard, similar to mathematicians. Our second class in the CS major, CS54 explores the theoretical underpinnings of computer science so if you continue with the CS intro that class will feel very different than CS51A. Other computer scientists work on more practical applications. They might examine questions related to hardware or to software. They might work with psychologists and anthropologists to understand how computers affect individuals and societies. They might work with lawyers and policy makers to build secure systems and protect our information. Some of these subfields include programming while others don't. What CS51A will do is give you a fundamental understanding of computational thinking, of programming, of AI, and of ethical implications of using technology. But there's an exciting world out there of more and different CS ideas to explore.

Lecture 1: Introduction

▸ Introduction

▸ **Logistics**

▸ Intro to Python

Are there any questions? Now that we covered these ambitious and exciting learning goals, let's see how we will achieve them!

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.

A typical week in CS51A will have the lectures on Monday and Wednesday. One lab that is either on Monday or Tuesday. And a weekly assignment that is due on Sunday. The assignments will be mostly programming-based in Python and they will include a short reading on ethics.

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.

All the information that you need about our weekly meetings and deliverables can be found on the course website. In fact, these very slides are online now. Please make sure to check the course website regularly. We will post slides and notes before class and it's always a good idea to review them again after each class to make sure you solidify your understanding.

## 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](#).

The textbook we will use for this class is called How to Think Like a Computer Scientist: Interactive Edition. It's by Brad Miller and David Ranum, based on original work by Jeffrey Elkner, Allen B. Downey, and Chris Meyers. It is available online for free and it is interactive with questions that can check your understanding as you study.

## Our mentors
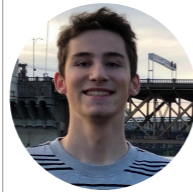


Jan Charatan
*he/him/his*

Matt Ivler
*he/him/his*

Michele Tang
*she/her/hers*

Nate Dailey
*he/him/his*

Aldo Ruiz Parra
*he/him/his*
BBICS mentor

Gisele Umutoni
*she/her/hers*
BBICS mentor

David D'Attile
*he/him/his*

Sarah Shader
*she/her/hers*

Shuyan Yan
*she/her/hers*

Beyond Dr. Dave, Professor Ye and me, we have a lot of support both within the class and the Department in general. First of all, we have an incredible team of TAs that will be assisting with the grading of the assignments, the Monday and Tuesday labs, and mentor sessions. Our CS students know how invaluable TAs are to our learning environment so please make sure you lean on them as well as all instructors for support.

## Slack Channels

▸ If registered, already invited to cs51a-spring2022 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

For the class, we will use the cs51a-spring2022 slack channel to answer questions off-class and to send announcements. If you are registered in CS51A, I have already invited you to this channel. Please let me know if you are facing any issues with it after class. You can post questions with your name known to everyone or anonymously.

There's also a departmental slack workspace with multiple channels that can help you socialize with other people in the department.

# Meet the 21-22 CS Liaisons!

Hello, world! I'm Gabriel Konar-Steenberg (he/him), junior CS major from Minneapolis. This is my second year as a liaison and a TA. I'm most interested in applying computing to societal problems; other interests include philosophy, environmental analysis, descending Wikipedia rabbit holes, and being outside. Feel free to contact me about anything at gmka2019@mymail.pomona.edu or on Slack!

Hi, I'm Sean O'Connor (he/him), a Senior and a nontraditional transfer student who previously worked in EMS for 5 years. Ask me about my research on privacy and surveillance, social justice and tech ethics, or surviving CS courses as a late convert from the humanities (swow2015@mymail.pomona.edu).

Hello, I'm Mercy Bickell (she/hers) and I'm a senior CS major, cog sci minor. I am a first time liaison and third time TA. I'm looking forward to meeting new and old students alike this year. I am particularly interested in NLP and the study of language in general. I also love video games and board games! You can contact me anytime on Slack or at mcba2018@mymail.pomona.edu.

Hi, I am Eryn Ma (she/her), a sophomore interested in the interdisciplinary study of CS. Feel free to chat with me about exploring intro courses and research as a new member of CS community, interdisciplinary studies like cognitive science, or transferring from online to in-person study! (ymac2020@mymail.pomona.edu)
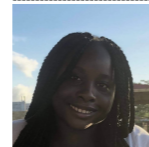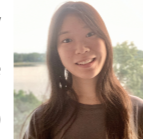
Hi my name is Maymuunah and my pronouns are she/her. I am a rising sophomore hoping to major in computer science with minor in either cognitive science or media studies. My email is mmqa2020@mymail.pomona.edu. Feel free to reach out!

Hey everyone! I'm Lucas Tiangco (he/him) '23. As someone who's experienced the stress of getting into the CS major at the last possible minute, I'd love to help anyone with any questions relating to CS major planning. I can also elaborate on CS research and my experiences as a TA. Of course, I'm also happy to chat about: your favorite movie; the one song that you've memorized the lyrics, melody, harmony, and backing vocals of; any interesting historical stories; and of course, your favorite way to eat a potato. If any of the above topics intrigue you, feel free to reach me at ldta2019@mymail.pomona.edu!

Outside the immediate CS51A team, our department is lucky to have six dedicated student liaisons. They act as intermediaries between students and faculty, organize social events, run surveys, and in general are an incredible resource that I urge you to talk to them if you have any ideas of events or you want to connect with the department and other students.

**POMONA COLLEGE**

# BLACK AND BROWN IN CS (BBICS)

*Providing support, resources, and encouragement to Black and Brown students in computer science*

**What we do?**
-----------------------------------------------
> Personal and academic mentoring
> Practice Interviews
> Monthly newsletter w/ shared resources
> Monthly speakers
-----------------------------------------------
contact | magali.ngouabou@pomona.edu

Finally, if you identify as a Black or a Brown person, I highly recommend you get in touch with BBICS, a student-run group that offers direct mentoring and many special events. Their contact person is Magali and she is incredible. Two of our TAs, Aldo and Gisele, are also BBICS mentors.

Sakai survey is due this Friday at 5pm

▸ Getting to know you

▸ Respond to questions you feel comfortable. Instructors have provided their responses to the questions in return.

To know you a bit better, we have released a Sakai survey named "getting to know you" on Sakai. Please fill it by this Friday afternoon.

## 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.

The grade calculation breakdown is as follows: 40% for assignments and labs, 35% for two midterms, 20% for the final, and 5% for participation. We typically don't accept late work as this class is fast-paced so you will want to keep up on a weekly-basis with the material we cover. Please keep in mind that both midterms and the final will be paper-based, even when writing code. You are expected to attend all lectures and labs and ask questions. The participation grade will include one mid-semester group presentation on a topic you choose on ethics. The readings we will be providing on a weekly basis can be a source of inspiration but feel free to also investigate topics you want.

## 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.

Since one of goals in this class is to make you confident programmers, we want you to learn how to work by yourself on a problem before seeking help. Please mindful of online resources: better stick to the course website, Python's official documentation, and textbook. Programming is not a solitary activity, in reality we often collaborate with others when we code. But we want you to be careful when talking with peers. Please don't look at each other's screens and work on each other's computers. It's OK to brainstorm on the whiteboard or a piece of paper together, but the solution and code should be your own. Finally, TAs are not there to give you answers but to help you reach your own answers. Since they are students themselves, please respect their time and don't ask them to work outside their designated mentor session time. Instead, use our slack channel.

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.

So how do you succeed in this course? It's actually really simple. Come to class and lab and ask questions throughout. If you have unanswered questions, email us or post them on Slack. I want you to take notes of confusing things or questions but please don't write down everything. This is why we have slides and notes. Finally, if you are confused or struggling within this class, in college, or life in general, please let one of the instructors or the TAs know. We are here to help.

Lecture 1: Introduction

▸ Introduction

▸ Logistics

▸ Intro to Python

Are there any questions about the class logistics or anything else? We are now ready to start talking about Python, the programming language we will be encoding our algorithms.

## 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.

The programming language we will be learning is Python. Python is an example of a high-level language; other high-level languages you might have heard of are C++ and Java. As you might infer from the name high-level language, there are also low-level languages, sometimes referred to as machine languages or assembly languages. Machine language is the encoding of instructions in binary (that is 0s and 1s) so that they can be directly executed by the computer. Loosely speaking, computers can only execute programs written in low-level languages. To be exact, computers can actually only execute programs written in machine language. Thus, programs written in a high-level language have to be processed before they can run. This extra processing takes some time, which is a small disadvantage of high-level languages. However, the advantages of high-level languages are enormous. First, it is much easier to program in a high-level language. Programs written in a high-level language take less time to write, they are shorter and easier to read, and they are more likely to be correct. Second, high-level languages are portable, meaning that they can run on different kinds of computers with few or no modifications. Low-level programs can run on only one kind of computer and have to be rewritten to run on another. Due to these advantages, almost all programs are written in high-level languages.

## 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")
```

This is an example of a low-level language code (top) being compiled by the high-level language C (bottom). You can see that a high-level language is much more readable and compact and resembles English a lot. Low-level code in contrast is very hard for humans to parse and is much lengthier.
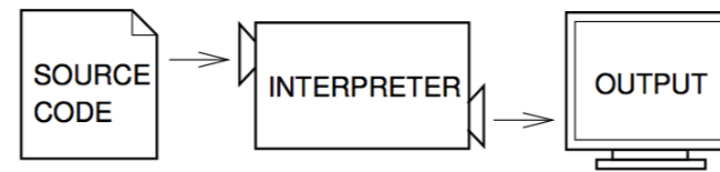
## Interpreters vs compilers



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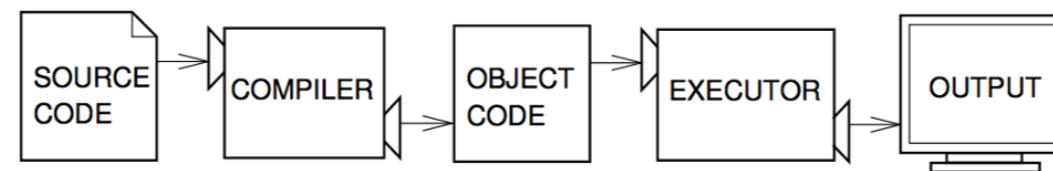


Figure 1.2: A compiler translates source code into object code, which is run by a hardware executor.

Two kinds of programs process high-level languages into low-level languages: interpreters and compilers. An interpreter reads a high-level program and executes it, meaning that it does what the program says. It processes the program a little at a time, alternately reading lines and performing computations. In contrast, a compiler reads the program and translates it completely before the program starts running. In this case, the high-level program is called the source code, and the translated program is called the object code or the executable. Once a program is compiled, you can execute it repeatedly without further translation. Broadly speaking, Python is considered an interpreted language.

## 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

The first way to interact with the Python interpreter is in shell model. In shell mode, you type Python expressions into the Python shell, and the interpreter immediately shows the result. The example below shows the Python shell at work. The >>> is called the Python prompt. The interpreter uses the prompt to indicate that it is ready for instructions. We typed 2 + 3. The interpreter evaluated our expression and replied 5. On the next line it gave a new prompt indicating that it is ready for more input. Working directly in the interpreter is convenient for testing short bits of code because you get immediate feedback. Think of it as scratch paper used to help you work out problems. Let's see how this works in practice but everything I will cover can be found in the slides.

## 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: %

Python makes for a great calculator. Just by opening the python shell, we can calculate the results of additions, subtractions, multiplications, divisions, powers, and remainders.

## 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
```

Let's do some basic math calculations that test addition, subtraction, multiplication, division, sub

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.

What is operator precedence? Python follows the normal operator precedence you're used to for math: things in parentheses get evaluated first, ** is next followed by %, * and / finally, - and +.

## Why are these different?

```
>>> 4+4
8
>>> 20/4
5.0
```

▸ "True division" always results into a floating-point number or float.

You might have noticed that when we added, subtracted, and multiplied two integer numbers the result was an integer, but when we divided two integers we received a decimal number. This is because in Python, division is "true". As we are used to 3/2 being 1.5, then 20/4 is also written in decimal form as 5.0. These numbers are known as floating point numbers or floats.

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.

When working with numbers, Python has two main numeric types: integers which can be negative, positive numbers, or zero and floating-point decimal numbers which are known as floats. Every value in Python has an associated type and for now we will only examine ints and floats.

## 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 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`.

Let's look into some basic definitions. The Python interpreter executes statements. A statement is an instruction that Python can execute, that is it is a command of what the interpreter should do. That means that a program is just a sequence of statements separated by new lines, although some statements might take multiple lines. In contrast, an expression is a combination of values and operators (this is a partial definition). Expressions need to be evaluated by the interpreter into a single value. Everything we have seen so far has been an expression, e.g., the expression 3+5, which consists of two values and an operator, 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.

## 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).

Let's assume you are having a COVID-safe BBQ party and you're trying to figure out how many hot dogs to buy. Here are some facts about your group of friends. Angie isn't a big fan of hot dogs, so she'll only eats 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 (assuming that if someone eats half a hot dog, we still have to count the whole thing). How did you do it? Could we do these calculations using the Python shell? Sure, but we would have to remember a lot of intermediate steps.

## Variables

‣ Variable: a name that refers to a value.

‣ A variable is essentially a storage for a value.
```
>>> angie = 1
>>> angie

1
```

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

‣ 
```
>>> angie = 2
>>> angle

2
```

‣ Expression is a combination of values, variables, and operators

    ‣ Still incomplete definition

A thing we can do to simplify our lives is to use variables. A variable is a name that we can use to refer to a value. You can think of a variable as a storage for a value. For example, I could have the variable angie and assign it the value 1 using the symbol = (note that it does not mean equality). I can actually change the value that angie holds from 1 to 2. By typing the variable, the interpreter will return the value it stores. Going back to the definition of expression, an expression is a combination of values, variables, and operators (but this is still an 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
```

Let's use the Python shell to calculate how many hot dogs we need for our BBQ party. We will use statements, variables, and expressions. We have six variables, angie, jasmine, chris, brenda, wenting, six variable assignments and expressions.

## 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
```

Did you notice that we got 12.5 hot dogs when we wanted to get back 13? This is because division in Python is real and when we divided the number of hot dogs that Brenda would have by 2 we got a float. We can fix this using integer division which is denoted by two slashes and which truncates the result by dropping the decimal numbers. For example, 11//2 =5, 10//3 =3, 11//3 =3, and 11.0//2 = 5.0 (because 11.0 is a float)

## How does integer division help us? $brenda//2=1$

```
>>> 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
```

So how can we use integer division to get 13 hot dogs? brenda//2 is 1 after all. What we will do is add one to force it to round up. And that works both for odds and even numbers.

## 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
```

So if I put everything together this is what I would have to type.

Naming variables

▸ Generally, you want to give good names 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`

In general, you want to give good names to variables that are self-explanatory. Please avoid x and y unless you represent x and y coordinates. The convention we will use is that all variable names should be lowercase and if we want to combine multiple words we will separate them with an underscore.
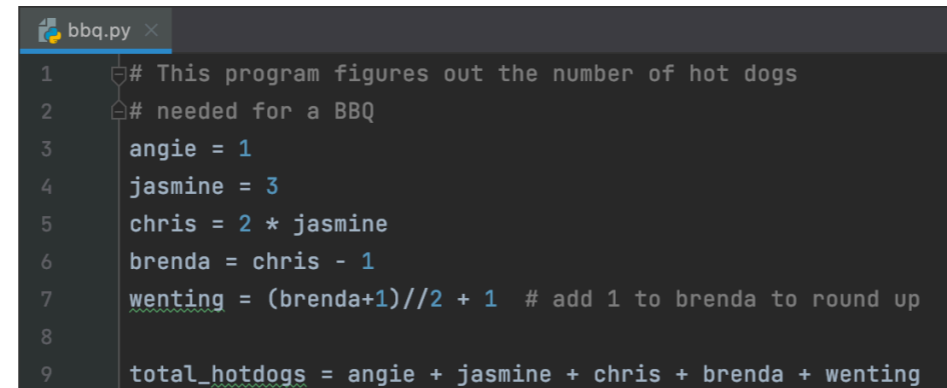
## Change of plans

▸ 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!

Let's assume now that Jasmine skipped breakfast and she's coming hungry to our BBQ wanting to have 4 hot dogs. To recalculate the total number of hotdogs we would need, we would need to type all the lines (except the first one) on the Python shell. This is getting annoying, we already had to do that once for integer division.

## Program mode

▸ Write source code in a .py and run it.

```
bbq.py ×
1    # This program figures out the number of hot dogs
2    # needed for a BBQ
3    angie = 1
4    jasmine = 3
5    chris = 2 * jasmine
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.

Luckily, we have a different way of interacting with the Python interpreter which is called program mode. Instead of entering one line at a time, we can write an entire program by placing lines of Python instructions in a file and then use the interpreter to execute the contents of the file as a whole. Such a file is often referred to as source code. For example, here I have a source code file named bbq.py and when I run it the interpreter reads and executes one line at a time. One of the differences with the interactive shell is that we don't get line-by-line feedback. We would need to "print" (to our screen not printer :) ) a variable to see its 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)

When you run a source code file, the Python interpreter interprets it and then ends the execution. If instead, we want to run our source code file AND continue having access to the variables defined in it, we would need to run the code a bit differently. We will right click anywhere on our file and then select "Run file in Python console". Now we can access all variables, e.g., angie, jasmine, etc. through our 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.

If you look at bbq.py you can see that we used whitespace without affecting how the code executes. In general whitespaces and blank lines make our code more readable.

We also used "comments" denoted by # which are just notes to ourselves and other programmers . Python ignores everything from the # to the end of the line. You can put comments on lines by themselves or if you have short comments, you can add them in line at the end of a line. Comments are VERY important and you will be required to put them in your programs for this course.

## PyCharm IDE

- IDE: Integrated Development Environment.

- 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.

Both of the ways that I showed you to interact with the Python interpreter, that is the interactive shell and program mode were done on PyCharm. PyCharm is an IDE, an Integrated Development Environments. IDEs make the life of aspiring developers such ourselves much easier. For example, PyCharm contains a text editor which we can use to edit and save our source code files such as bbq.py. It also contains tools for running our source code and for debugging it, that is find mistakes in it. All of our code will be organized in projects and we will be able to navigate through different projects and source code files. PyCharm also contains a Python Console which is the Python shell we talked about. The course website and first lab contain instructions on how to install PyCharm on your own machines. You can also configure and rearrange how PyCharm looks, giving it your personal style. You will see more about PyCharm in the first lab.

## Resources

▸  [Two AIs Talk About Why the Universe Exists. (GPT-3)](#)

▸  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.