CMPU 100 · Programming with Data

Expressions, Values, and Names

Class 2

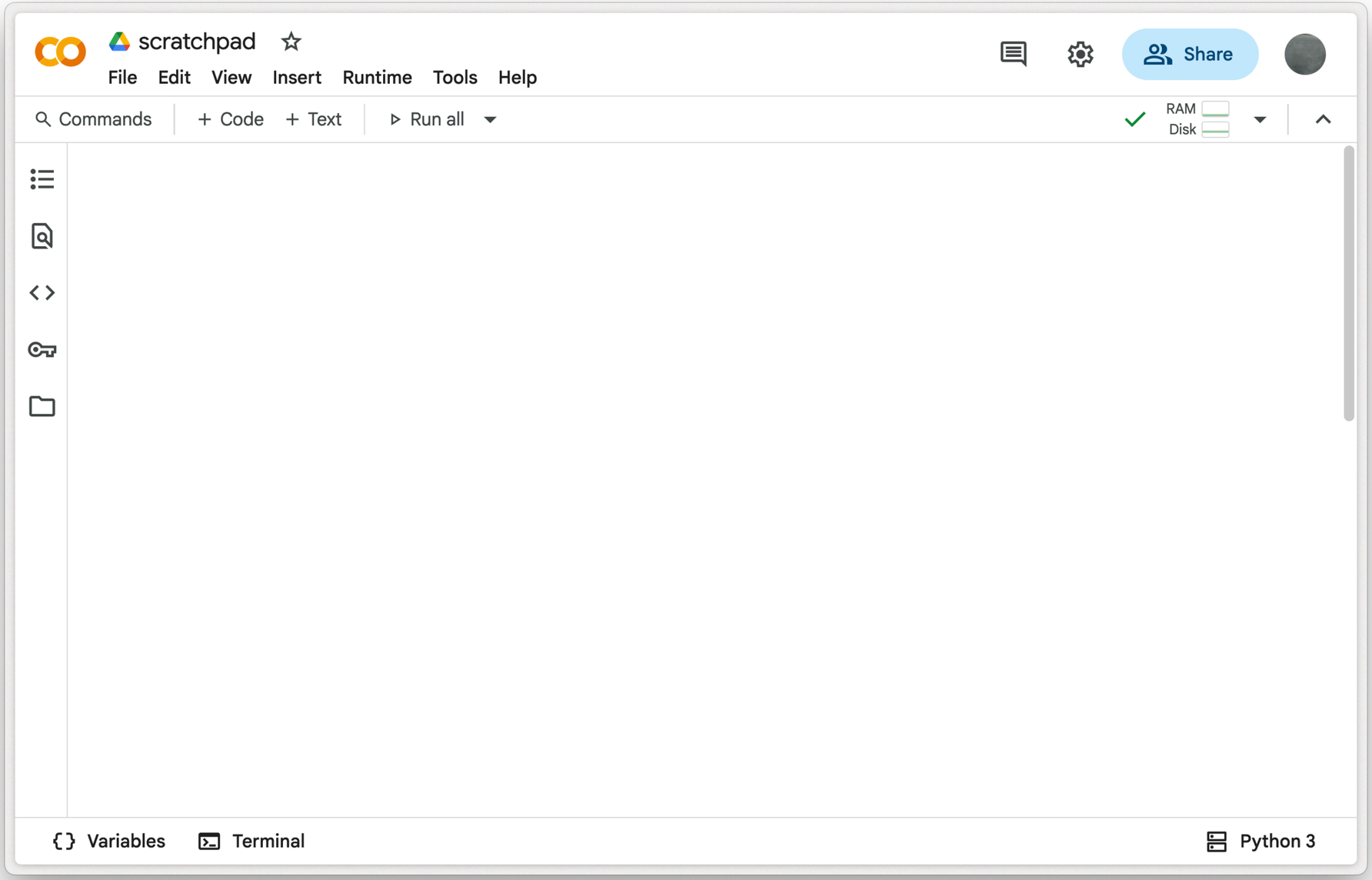
Where are we?

A *program* instructs a computer to do something.

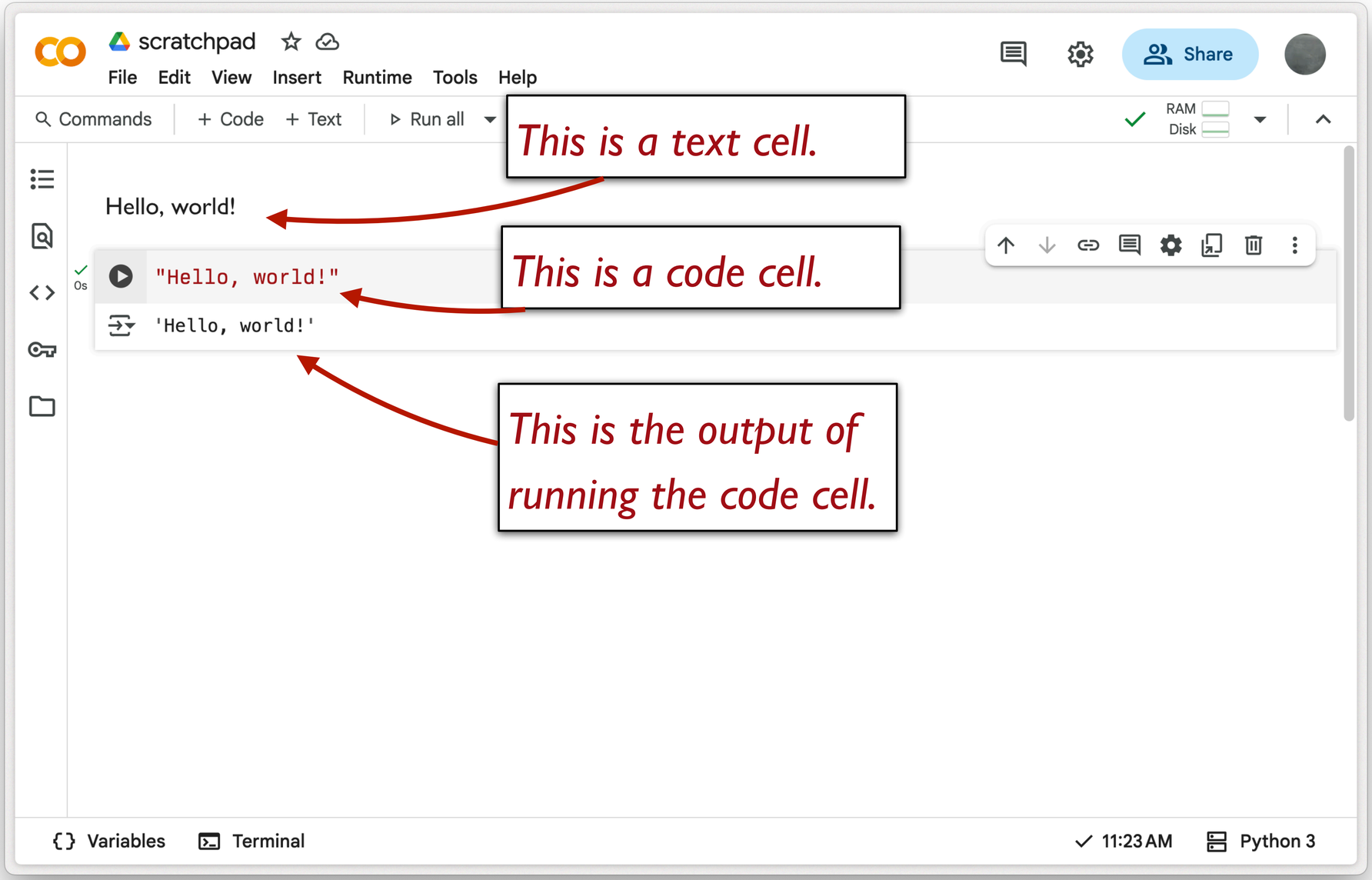
For the computer to carry out these instructions, they need to be *precise*.

But programs also need to be understood by people, so they need to be *readable*!

We write a program in a *programming language* and we run it in a *programming environment*.



colab.research.google.com


scratchpad
File Edit View Insert Runtime Tools Help
Share
Commands
+ Code
+ Text
Run all
RAM
Disk
This is a text cell.
Hello, world!
This is a code cell.
"Hello, world!"
0s
'Hello, world!'
This is the output of running the code cell.
Variables
Terminal
11:23 AM
Python 3

→ Notebook: *Warm-up*

Expressions

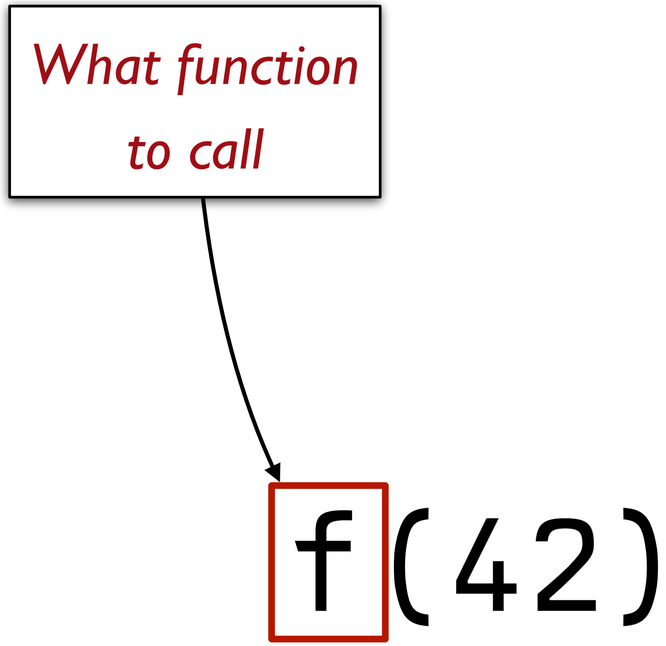
`(3 + 4) * (5 + 1)` is an *expression* – a computation that produces an answer.

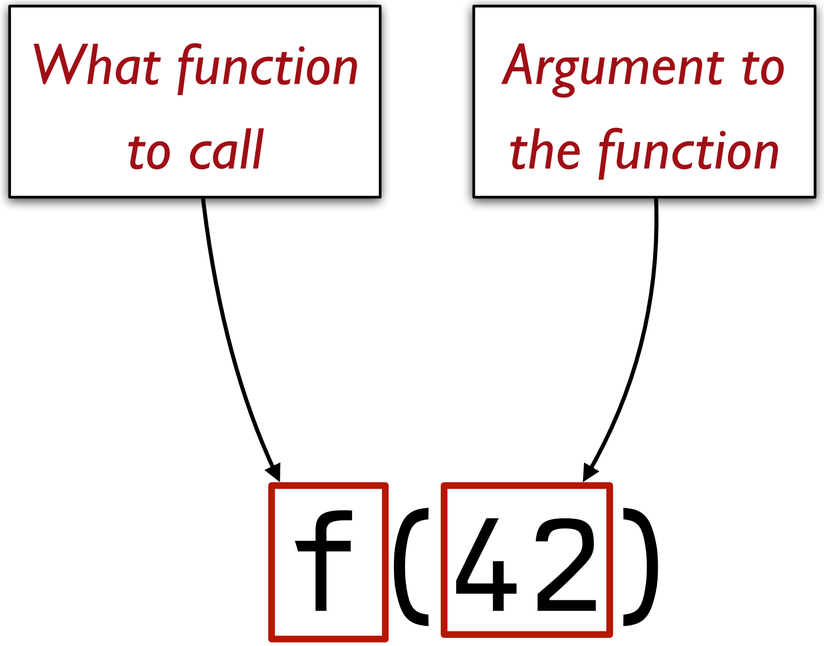
A program just consists of one or more computations you want to run.

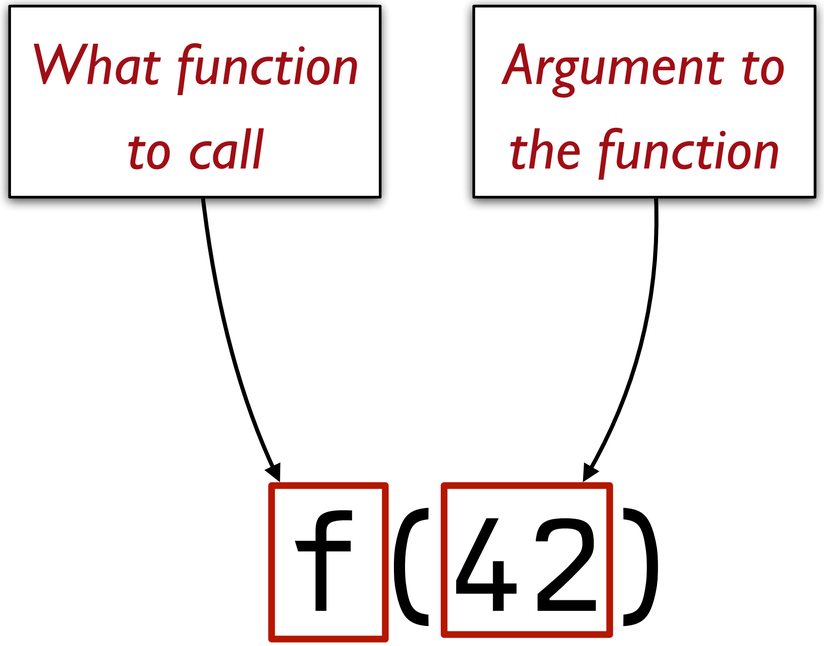
An individual number like `17` is a *value* (or *literal*); it can't be computed any further.

Call expressions

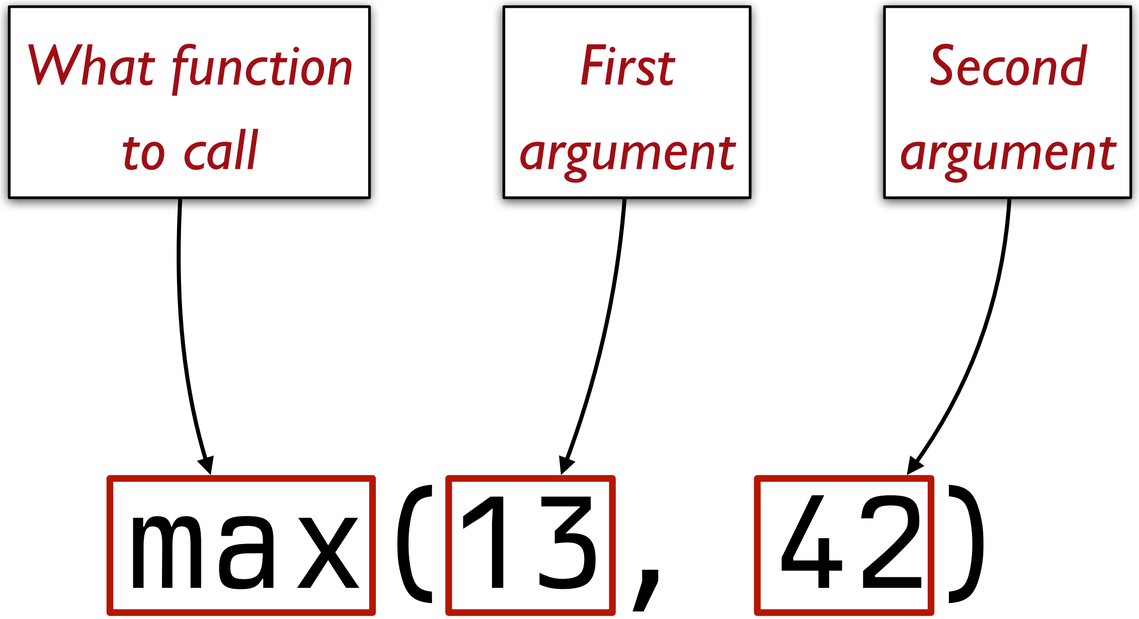
f(42)


What function to call
f(42)


What function to call
Argument to the function
f(42)



"Call f on 42."


What function to call
First argument
Second argument
max(13, 42)

→ Notebook: *Expressions*

Example: *Incidence of tuberculosis*

Morbidity and Mortality Weekly Report (*MMWR*)

Search

Tuberculosis — United States, 2021

Weekly / March 25, 2022 / 71(12);441–446

Print

Thomas D. Filardo, MD[1,2]; Pei-Jean Feng, MPH[2]; Robert H. Pratt[2]; Sandy F. Price[2]; Julie L. Self, PhD[2] (VIEW AUTHOR AFFILIATIONS)

View suggested citation

Summary

What is already known about this topic?

The number of reported U.S. tuberculosis (TB) cases decreased sharply in 2020, possibly related to multiple factors associated with the COVID-19 pandemic.

What is added by this report?

Reported TB incidence [illegible] rom 2.2 during 2020 to 2.4 during 2021 but was lo[illegible]s occurred among both U.S.-born and non–U.S.-born persons.

What are the implications for public health practice?

Factors contributing to changes in reported TB during 2020–2021 likely include an actual reduction in TB incidence as well as delayed or missed TB diagnoses. Timely evaluation and treatment of TB and latent tuberculosis infection remain critical to achieving U.S. TB elimination.

What is *incidence*?

TABLE 1. Tuberculosis disease case counts and incidence, by U.S. state — 50 states and the District of Columbia, 2019–2021

Return

U.S. jurisdiction	No. of TB cases*			TB incidence†		
	2019	2020	2021	2019	2020	2021
Total	**8,900**	**7,173**	**7,860**	**2.71**	**2.16**	**2.37**
Alabama	87	72	92	1.77	1.43	1.83
Alaska	58	58	58	7.91	7.92	7.92
Arizona	183	136	129	2.51	1.89	1.77
Arkansas	64	59	69	2.12	1.96	2.28
California	2,111	1,706	1,750	5.35	4.32	4.46
Colorado	66	52	58	1.15	0.90	1.00
Connecticut	67	54	54	1.88	1.50	1.50
Delaware	18	17	43	1.84	1.71	4.29
District of Columbia	24	19	19	3.39	2.75	2.84

TABLE 1. Tuberculosis disease case counts and incidence, by U.S. state — 50 states and the District of Columbia, 2019–2021

Return

U.S. jurisdiction	No. of TB cases*			TB incidence†		
	2019	2020	2021	2019	2020	2021
Total	**8,900**	**7,173**	**7,860**	**2.71**	**2.16**	**2.37**
Alabama	87	72	92	1.77	1.43	1.83
Alaska	58	58	58	7.91	7.92	7.92
Arizona	183	136	129	2.51	1.89	1.77
Arkansas	64	59	69	2.12	1.96	2.28
California	2,111	1,706	1,750	5.35	4.32	4.46
Colorado	66	52	58	1.15	0.90	1.00
Connecticut	67	54	54	1.88	1.50	1.50
Delaware	18	17	43	1.84	1.71	4.29
District of Columbia	24	19	19	3.39	2.75	2.84

† Cases per 100,000 persons using midyear population estimates from the U.S. Census Bureau. 2019 population estimates are based on the 2010 U.S. Census. 2020 and 2021 population estimates are based on the 2020 U.S. Census. https://www.census.gov/programs-surveys/popest/data/tables.html

TABLE 1. Tuberculosis disease case counts and incidence, by U.S. state — 50 states and the District of Columbia, 2019–2021

Return

U.S. jurisdiction	No. of TB cases*			TB incidence†		
	2019	2020	2021	2019	2020	2021
Total	**8,900**	**7,173**	**7,860**	**2.71**	**2.16**	**2.37**
Alabama	87	72	92	1.77	1.43	1.83
		58	58	7.91	7.92	7.92
	3	136	129	2.51	1.89	1.77
		59	69	2.12	1.96	2.28
	1	1,706	1,750	5.35	4.32	4.46
Colorado	66	52	58	1.15	0.90	1.00
Connecticut	67	54	54	1.88	1.50	1.50
Delaware	18	17	43	1.84	1.71	4.29
District of Columbia	24	19	19	3.39	2.75	2.84

† Cases per 100,000 persons using midyear population estimates from the U.S. Census Bureau. 2019 population estimates are based on the 2010 U.S. Census. 2020 and 2021 population estimates are based on the 2020 U.S. Census. https://www.census.gov/programs-surveys/popest/data/tables.html

Let's use Python to validate these values. We'll check the 2020 and 2021 incidence for the US as a whole.

Did tuberculosis become 10% more common in the United States in 2021?

2.16 → 2.37

2020 2021

+10%

in a single year

→ Notebook: *Checking the CDC's number*

Types of values

What we saw were two different *data types* used for numbers in Pythons:

Integers

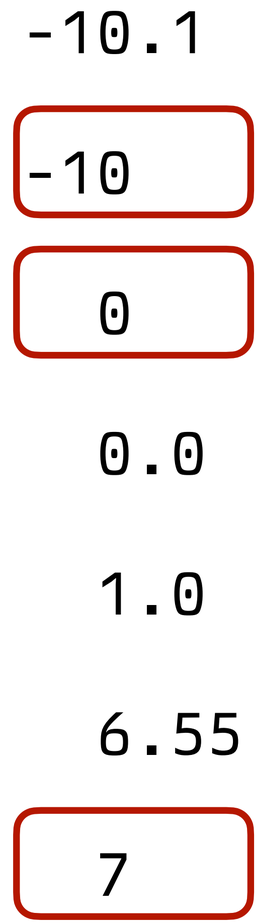
Floating-point numbers

```
-10.1

-10

  0

  0.0

  1.0

  6.55

  7
```

What we saw were two different *data types* used for numbers in Pythons:

Integers are whole numbers

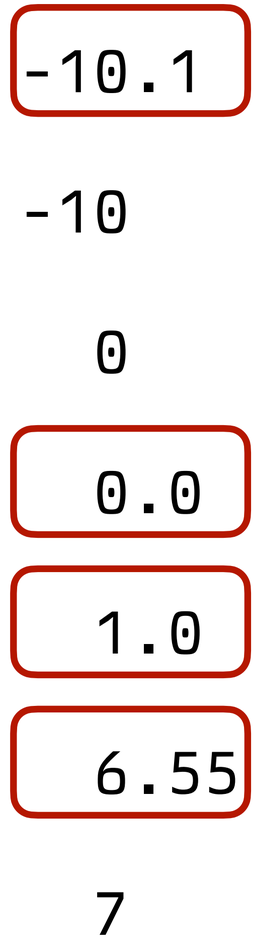
Floating-point numbers



What we saw were two different *data types* used for numbers in Pythons:

Integers are whole numbers

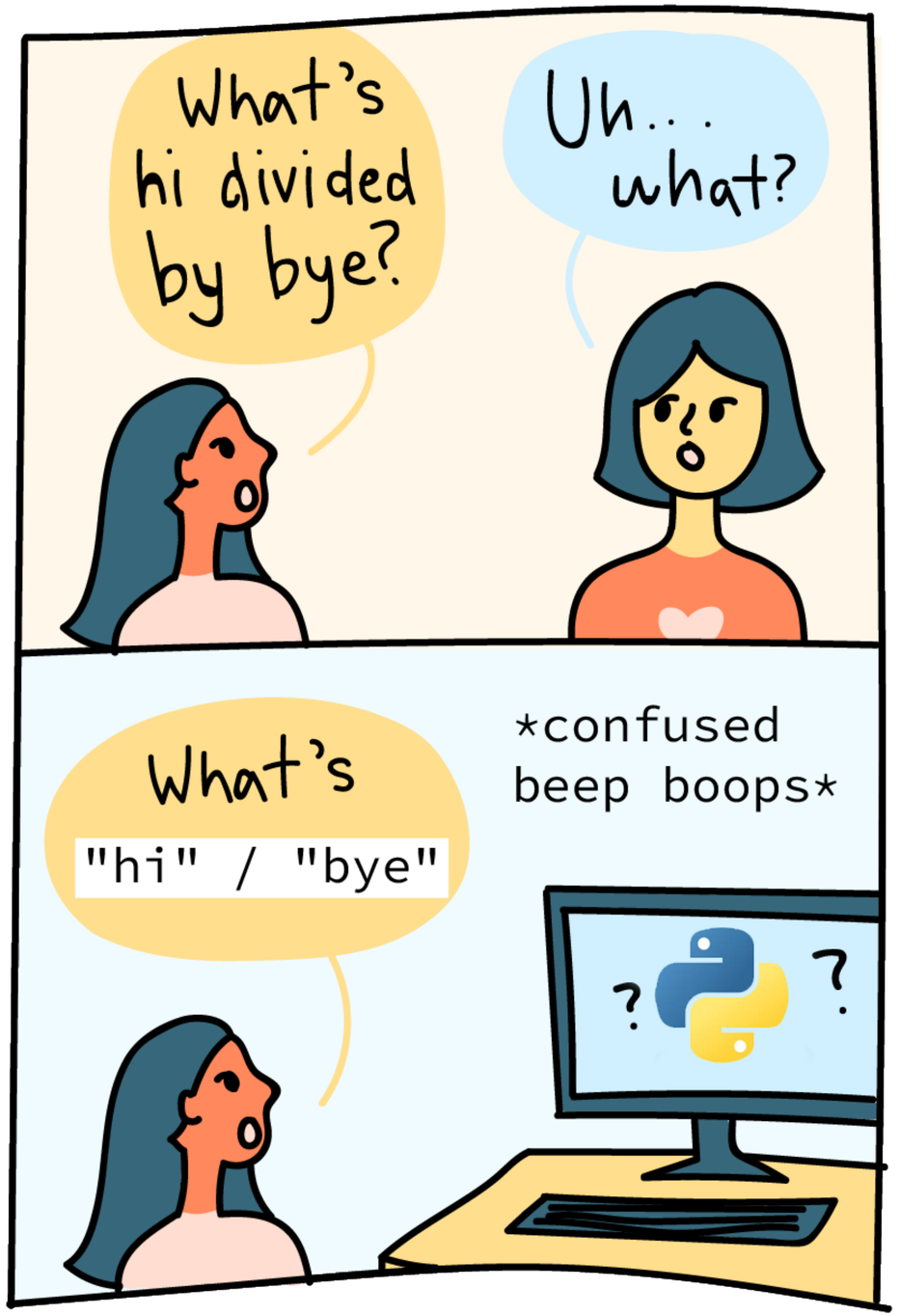
Floating-point numbers have a decimal point



→ Notebook: *Types of values*

Working with different types of values

Operations may only work on certain types of data!



Same operator, different types

Before we run them, write down what you think each of these will evaluate to:

```
3 + 4
```

```
"3" + "4"
```

```
3 * 4
```

```
"3" * 4
```

```
max(3, 4)
```

```
max("three", "four")
```

→ Notebook: *Working with different types of values*

Evaluation

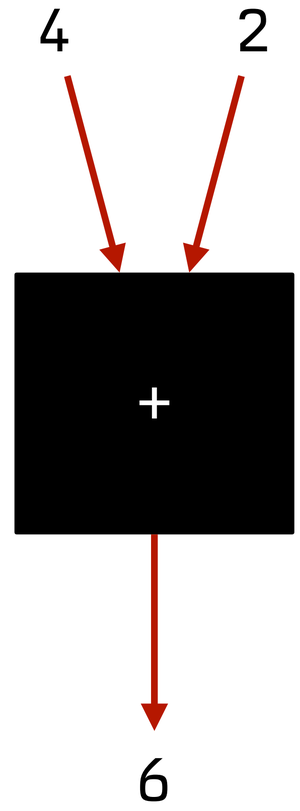
How does something like `(4 + 2) / 3` work?

What is the operator `/` dividing?

Shouldn't `/` expect two numbers?

Even though `(4 + 2)` isn't a number, it's an expression that *evaluates* to a number.

This works for all data types, not just numbers!



What does this evaluate to?

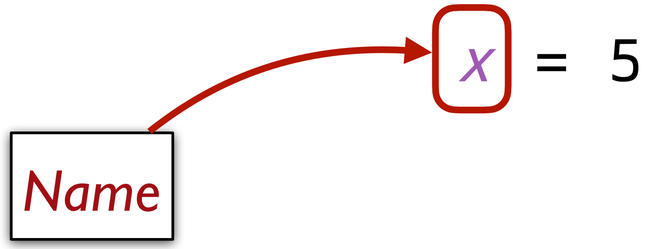
```
min(abs(max(-1, -2, -3, min(4, -2))), max(5, 100))
```

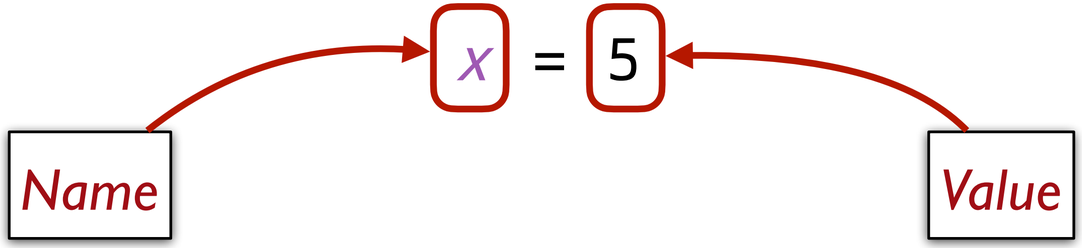
→ *?*

→ Notebook: *Try it yourself: nested expressions*

Names

$$x = 5$$


x
=
5
Name


x
=
5
Name
Value

$x = 5$

The name **x** *is bound to the value* **5**, *like putting a baggage tag on a suitcase.*

```
x = 5

y = 1 + 2 * 3 - 8 / 2
```

First Python evaluates the right-hand expression…

```
x = 5

y = 1 + 2 * 3 - 8 / 2
→ y = 1 + 6     - 8 / 2
```

First Python evaluates the right-hand expression…

```
x = 5

  y = 1 + 2 * 3 - 8 / 2
→ y = 1 + 6     - 8 / 2
→ y = 7         - 4
```

First Python evaluates the right-hand expression…

```
    x = 5

    y = 1 + 2 * 3 - 8 / 2
→   y = 1 + 6     - 8 / 2
→   y = 7         - 4
→   y = 3
```

First Python evaluates the right-hand expression…

```
x = 5

y = 1 + 2 * 3 - 8 / 2
→ y = 1 + 6     - 8 / 2
→ y = 7         - 4
→ y = 3
```

First Python evaluates the right-hand expression…

…then it binds the name y to the resulting value.

Assignment statements are *not* mathematical equations.

If you write

```
3 = x
```

Python gives a syntax error because it thinks you're trying to redefine what "3" means.

$x = 5$

```
x = 5
```

There's no output from assigning a name to a value.

```
x = 5
```

Directory

Name	***Value***
x	5

There's no output from assigning a name to a value.

It has the side effect of associating the name with the value in the program directory.

Directory

Name	***Value***
x	5

```
x = 5
```

```
x
```

Directory

Name	***Value***
x	5

When you use the name later, Python looks it up in the directory and substitutes the value it finds.

x = 5

x

→ 5

Directory

Name	***Value***
x	5

When you use the name later, Python looks it up in the directory and substitutes the value it finds.

A name can only be bound to a single value at one time.

A name can only be bound to a single value at one time.

$x = 2$

A name can only be bound to a single value at one time.

$x = 2$

A name can only be bound to a single value at one time.

$x = 2$

$x = x + 1$

A name can only be bound to a single value at one time.

```
x = 2
x = x + 1
```

Don’t delete cells defining names you want to use.

Don’t use names *above* the cell with the assignment definition.

Notebooks should be a paper trail. Each cell is a record of what you’ve done so far.

→ Notebook: *Names*

Concept check

Quick check

For each of the cells below, what is the output?

```
side_length = 5
area = side_length ** 2
side_length = side_length + 2
```

```
side_length
```

```
area
```

In Python, the = operator is assignment, not equality.
So, **x = y** is different from **y = x**.

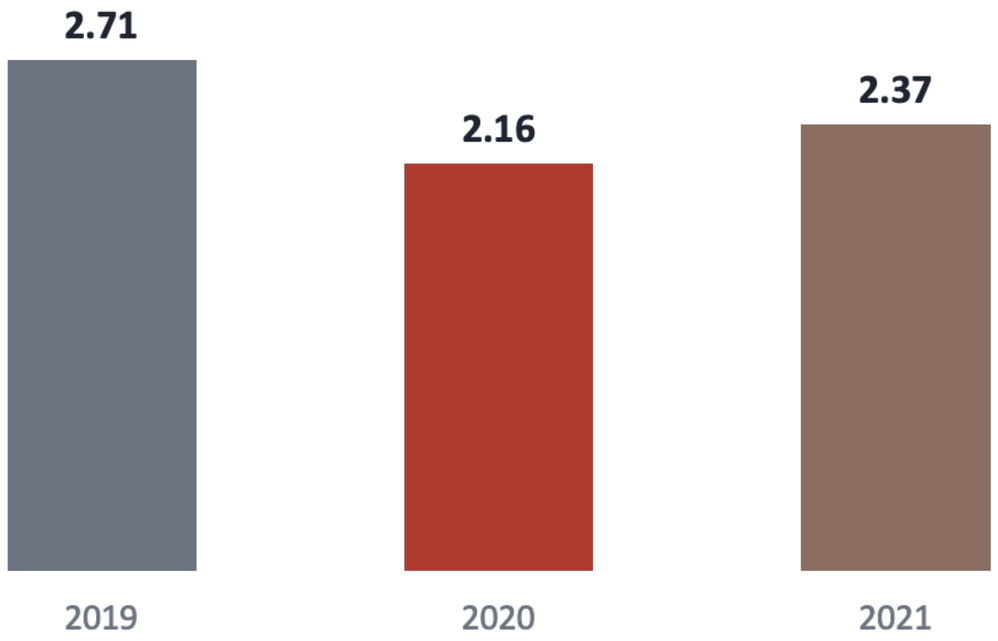
Here are the same lines in different order:

```
x = 5
x = y
y = x
y
```

```
x = 5
y = x
x = y
y
```

What does each one do?

→ Notebook: *Rebinding*



The rise is real arithmetic. But look what it's a rise from.

Incidence fell about 20% in 2020 – the largest one-year drop in decades – and recovered only part of the way.

2021 is still well below 2019.

Was there *less* tuberculosis in the US in 2020?

Or did we *find* less tuberculosis in 2020?

Our data can't tell us. The case count isn't a count of people who had TB; it's a count of the people who were diagnosed and reported – in a year that hospitals were overwhelmed due to COVID.

The uptick was actually the start of a trend

By 2023, US tuberculosis incidence was above where it had been before the pandemic. The blip we computed turned out to be the beginning of a real rise – which is why asking how the data was made was worth doing.

Writing code for people to read

“Programs must be written for people to read, and only incidentally for machines to execute.”

Hal Abelson & Gerald Sussman with Julie Sussman, *Structure and Interpretation of Computer Programs*, 1979

Names are important!

Can you guess what this code does?

```
y = (x + 459.67) * 5/9
```

Names are important!

Can you guess what this code does?

```
y = (x + 459.67) * 5/9
```

```
temp_kelvin = (temp_fahrenheit + 459.67) * 5/9
```

Comments are used to explain what code does.

Good programmers write code that is *self-evident*
and use comments only where necessary.

→ Notebook: *Writing code for people to read*

Acknowledgments

This class incorporates material from:

Kathi Fisler, Brown University

Jason Waterman, Vassar College

Data 6, University of California, Berkeley (CC BY-NC-SA)