CS 61A SU26

Lecture 01: Welcome, Coding Environment, Functions, and Exceptions I

June 22, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

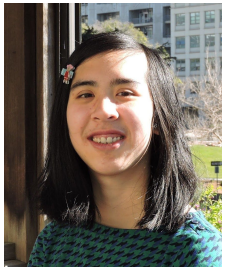
Welcome

30 min

- Introductions
- Pre-semester survey
- About the course
- Course materials
- Syllabus, logistics, and support
- Questions or concerns? Here's how to let us know
- AI
- *If time:*
  - *Computing in the news*
  - *What do you want to build?*

Introductions (1 of 2)

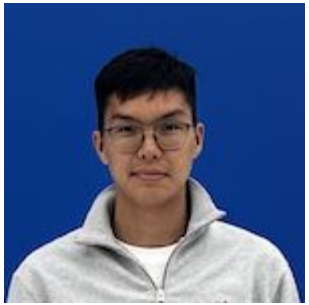
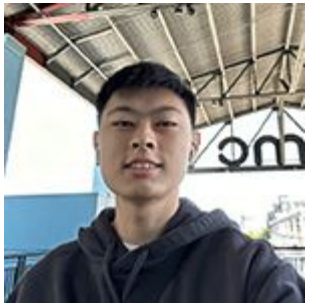
Hey there, I'm Rebecca (she/her)!



- **Education:** Cal BS EECS '25, MS EECS '26 (go bears 🐻)
- **Teaching:**
  - DATA 188 (Intro to Deep Learning)
  - DATA C88C (Computational Structures in Data Science) 5x
  - DATA 101 (Data Engineering)
  - CS 61A (teaching/academic intern)
  - Computer Science Mentors 4x (volunteer tutoring)
- **Professional (all SWE internships):**
  - Databricks (also incoming full-time)
  - Stripe
  - Bloomberg 2x
- **Hobbies/interests:** Watching movies/TV shows, reading, hiking, plushies
- **Website:** phrdang.github.io

TAs!





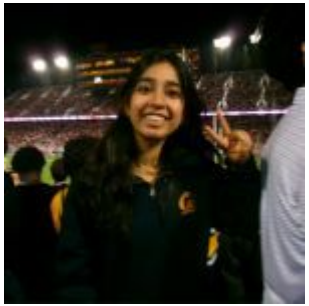


Tutors!







Introductions (2 of 2)

Turn to someone next to you and introduce yourselves!

- Name
- Year
- Major
- 1 thing you're excited about this summer
- 1 thing you're nervous about this summer

go.cs61a.org/presem

Please fill out our pre-semester survey!

(Part of Lab 0, so might as well do it now)

About the course

CS 61A: The Structure and Interpretation of Computer Programs

- You will learn 3 programming languages: Python, Scheme, and SQL
- … but this is *not just* an introductory programming course
- Key ideas: Abstraction, problem solving, managing complexity, and interpreting programming languages
- … in other words, *how to think like a computer scientist*

How to succeed? (see also: Advice from past students)

- Attend and actively participate in all synchronous parts of the course (lecture, lab, discussion)
- Do the assignments yourself and come to OH/ask questions on Ed/talk to classmates if you get stuck **(productive struggle)**
- Exams: Do problems from past exams

Course Materials

(Almost) everything is on our course website, cs61a.org

- Lecture recordings: bCourses > Media Gallery (~1 day delay)
- Q&A: edstem.org
- Submitting assignments: Gradescope
- Textbook: Composing Programs

IMPORTANT: You are solely responsible for knowing the syllabus/course policies, assignment deadlines, and checking your email and Ed for announcements.

Today's lecture is NOT comprehensive of everything you need to know.

Syllabus, logistics, and support (1 of 3)

Syllabus: cs61a.org/articles/about-61a/

- **New this summer:**
  - We are 100% in-person (decision from the chairs)
  - Weekly Computer-Based Testing Facility (CBTF) quizzes
    - *Why? Frequent, small-stakes testing > single high-stakes test*
  - Grading policy: ⅔ exams, ⅓ everything else
    - *Why? AI coding tools make take-home assignments a noisy signal of students' understanding and skill*
- **New since our welcome Ed announcement:**
  - 61a-bot enabled on HW
  - Orientation quiz release date moved to Tue 6/23

Syllabus, logistics, and support (2 of 3)

- **OH:** cs61a.org/office-hours/ (calendar may change, check before going)
- **DSP accommodations:** Please submit your letter of accommodation ASAP
- **Lab/discussion excused absence requests:** Email cs61a@berkeley.edu
- **Extensions:** Submit request on go.cs61a.org/extensions (Flextensions)
- **Coming soon:** Exam prep tutoring sections

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

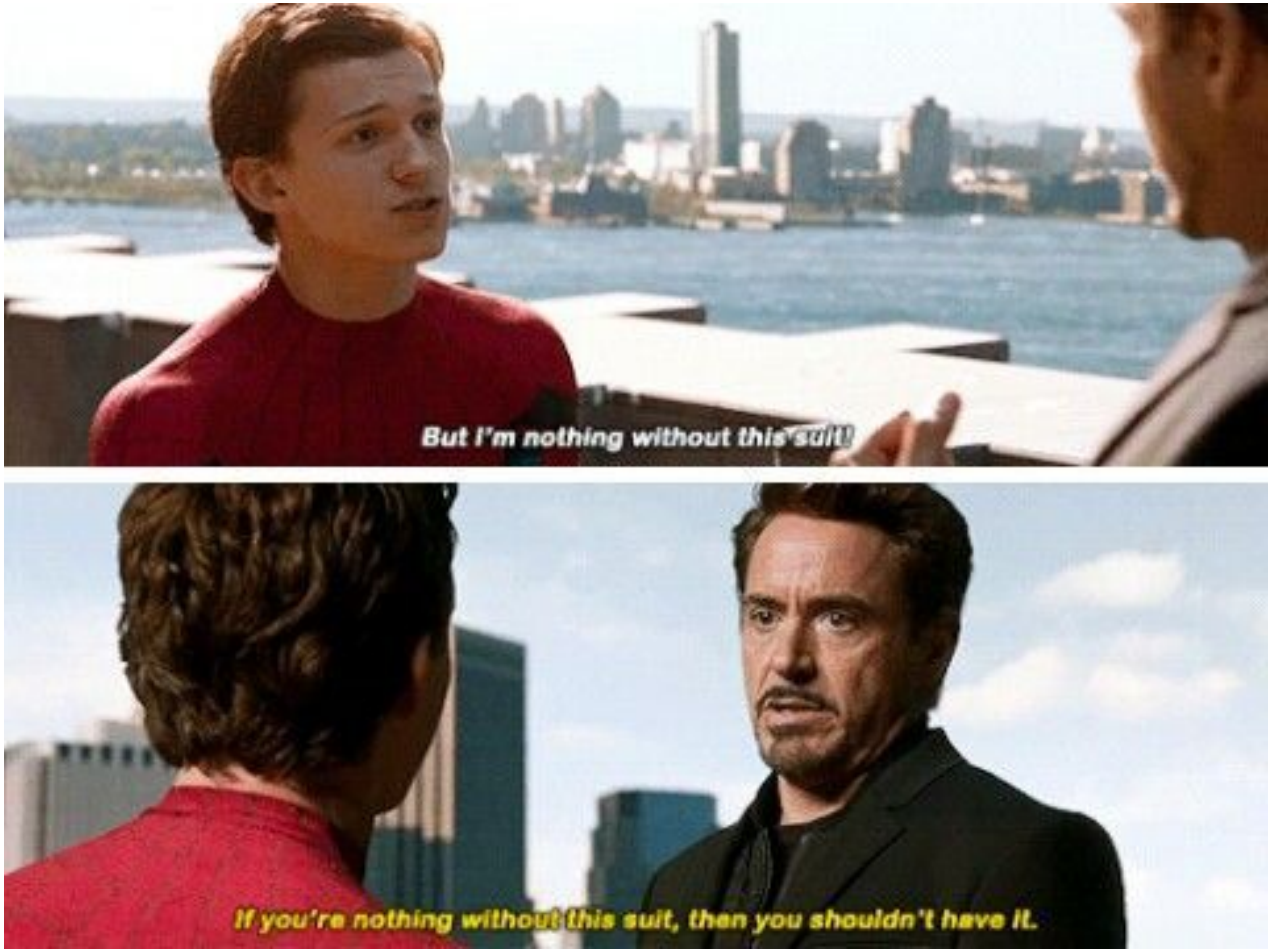
If you need to request an excused absence from lab or discussion section, how do you let us know?

Questions or concerns? Here's how to let us know

- **Is your question about the course content or logistics?** → Ed or OH
- **Are you requesting an extension?** → Flextensions
- **Are you requesting an excused absence from lab/discussion?** → cs61a@berkeley.edu
- **Is your question about DSP accommodations?** → cs61a@berkeley.edu
- **Is your question about an extenuating circumstance or sensitive, but you're okay with admin TAs and the instructor seeing it?** → cs61a@berkeley.edu
- **Is your question only something that the instructor can answer or otherwise sensitive?**
  - **Do you want to be anonymous?**
    - **Yes** → go.cs61a.org/anon-feedback
    - **No** → rdang@berkeley.edu (me)

AI

- Our course AI policy TLDR:
  - Using AI to code for you ❌
  - Using AI as a documentation or learning resource ✅
- Why?
  - Yes, we know AI coding tools are widely used on the job today
    - But do you want to be the person who mindlessly vibecodes all day?
    - Do you want to be reliant on a corporation (which cares more about profit than you) for your thinking and skills?
    - How can you judge that AI output is **good and correct** without knowing at ***least*** the fundamentals?
  - Critical thinking, problem solving, design, communication, etc. are still important skills.
  - At the end of the day, ***you as the engineer are accountable*** for all AI outputs


But I'm nothing without this suit!
If you're nothing without this suit, then you shouldn't have it.

If time: Computing in the news

- **Note:** All computing in the news sections are not in scope for exams, but great to know about as a person!
- In Age of AI, World's Leading Deepfake Expert No Longer Trusts His Own Eyes (New York Times)
    - Featuring former UC Berkeley Professor Hany Farid
- Failing grades soar as professors see greater AI usage, dwindling math skills in UC Berkeley computer science classes (Daily Cal)
- Anthropic Pulls Its Most Powerful AI Models After U.S. Bars Foreign Access (TIME Magazine)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What do you want to build with your knowledge of CS?

Coding Environment

10 min

- File System
- Terminal
- Python Interpreter

File System

- File
- Directory (folder)
- Path: Location of a file or directory
  - Absolute: Your full address
  - Relative: "I'm living at the apartment 3 blocks north of here"

Demo: Terminal

- Command-line (text) based program to interact with your computer and file system
- Demo (Unix/Mac/WSL)
  - Parts of a terminal prompt
  - Up/down arrow for command history
  - `pwd`: print working directory
  - `ls`: list files and directories
  - `cd`: change directory
    - `cd ..` → go to parent directory
    - `cd ~` → go to home directory
  - `touch`: create a file
  - `mkdir`: make directory
  - `rm`: remove a file or directory **(⚠ danger!)**
- **[out of scope]** Customize your prompt with Starship

Demo: Python Interpreter

- How to open: Install Python 3, then run `python3`
- Terminal vs. Interpreter
  - Terminal: Run commands
  - Interpreter: Run Python code
- How to exit: `exit()`, `quit()`, or Ctrl + D
- Interpreter vs. IDEs (Visual Studio Code)
  - Interpreter: Temporary, demoing code
  - IDE (integrated development environment): Permanent, writing longer snippets of code (in a file), more features for software engineering

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

What should you do here?

Functions

30 min

- Primitive data types
- Expressions
- Names and Assignment
- Functions
- Print vs. Return

Primitive data types

- Integer, ex: -5, 0, 100
- Float, ex: -1.1, 0.0, 43.9235
- String, ex: `'hello world'` or `"hello world"`
  - Multiline strings or comments: Triple single or double quotes
- Boolean, ex: `True` or `False`

Demo: Expressions

Way to combine data types. Python automatically computes or **evaluates** expressions. # denotes a single line **comment** in Python.

```
>>> (2 + 2) * 5
20
>>> 5 ** 2  # exponent
25
>>> 10 / 3  # true division
3.3333333333333335
>>> 10 // 3  # floor division
3
>>> 10 % 3  # modulo operator
1
```

```
>>> 'hello'
'hello'
>>> 'hello' + 'world'
'helloworld'
>>> 'a' * 5
'aaaaa'
```

Demo: Names and assignment

What if we want to give a label to a particular value or save it for later use?

We can bind a value to a **name** (often called a **variable**, although names can be used for other things).

```
>>> pi = 3.1415926
>>> pi
3.1415926
>>> # multiple assignment
>>> x, y = 1, 3
>>> x
1
>>> y
3
```

```
>>> name = 'Rebecca'
>>> f'Hello {name}!'  # f-string
'Hello Rebecca!'
>>> # import statement
>>> from operator import mul
>>> mul(10, 5)
50
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Names and assignment: What would Python display?

Demo: Functions (1 of 2)

- What if we want to abstract away a set of operations so that we can reuse it? We can define our own **functions**!
- A function can take 0 or more inputs or **parameters**, and return 1 or more outputs or **return values**.
- The value you pass into a parameter is called an **argument** (e.g. below, 5 is the argument).
- **Calling** (or using) a function involves a **call expression**: The function name and its argument(s) in parentheses, separated by commas.

```
>>> from math import pi
>>> def circle_area(radius):
...     return pi * (radius ** 2)
>>> circle_area(5)
78.53981633974483
```

Demo: Functions (2 of 2)

- Functions can **return multiple values** separated by commas
  - Technically, it's a single tuple, but we'll get to that later...
- Functions can have **default arguments**
- **Type hints** which tell the interpreter and programmer the data type of each parameter and return value
- **Docstrings** are multiline strings right below the **function signature** that document the behavior of the function
  - They may contain **doctests**, or code that show how the function is supposed to behave

```
def circle_area(radius: float = 1) -> float:
    """Returns the area of a circle
    with the given radius.

    >>> from math import pi
    >>> circle_area()
    3.141592653589793
    >>> circle_area(5)
    78.53981633974483
    """
    return pi * (radius ** 2)
```

Demo: Builtins

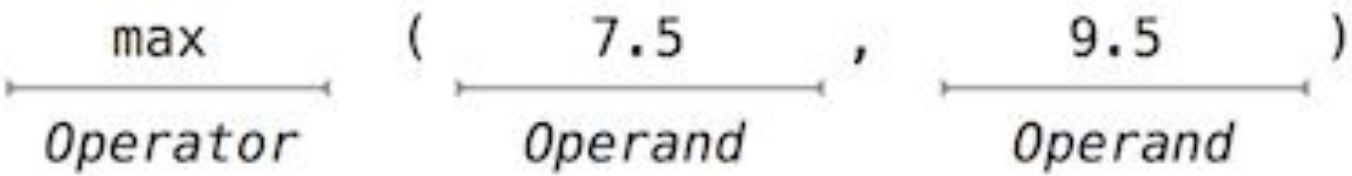
Python has several **built-in functions** you can use without importing.

```
>>> max(1, 2, 3)
3
>>> min(1, 2, 3)
1
>>> len('hello')
5
>>> sum(1, 2, 3)
6
>>> print('hello world')
hello world
```

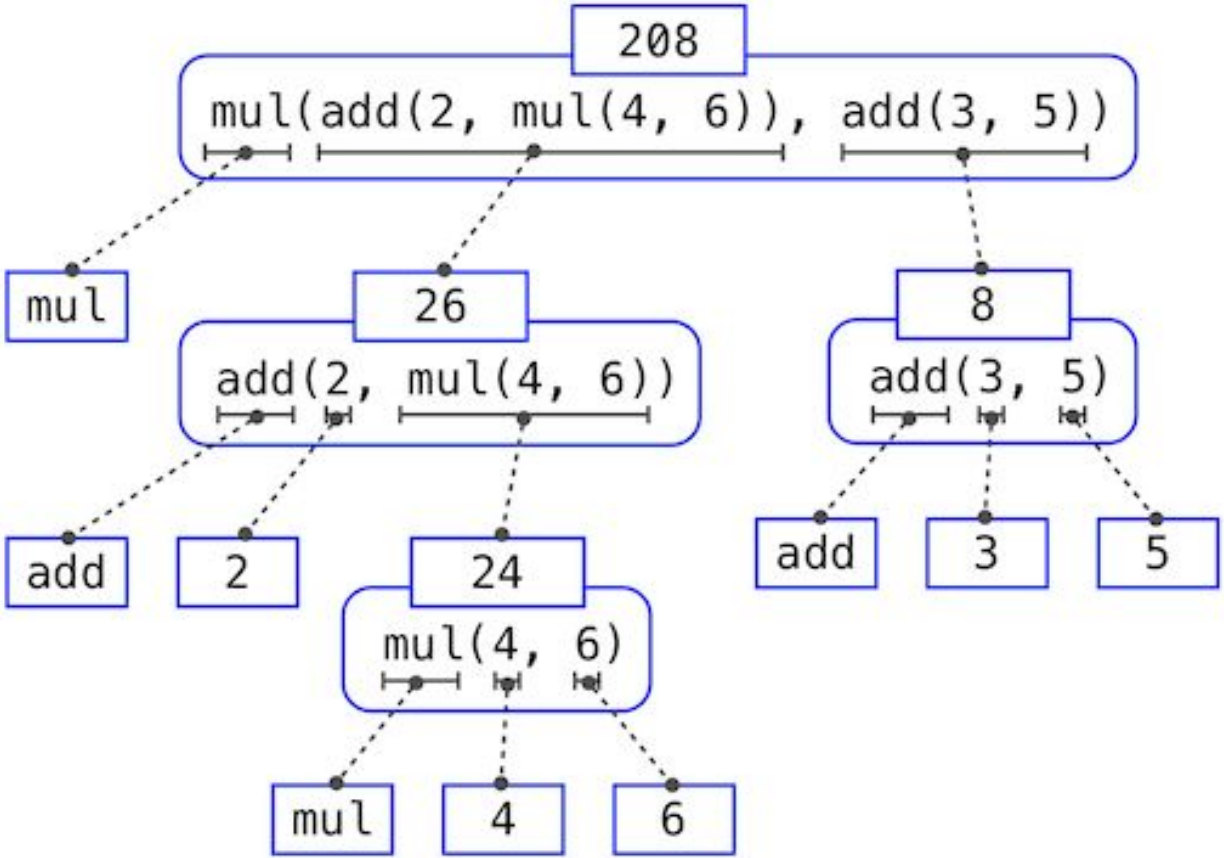
Functions: Call expressions (1 of 2)

Call expression order of evaluation:

1. **Evaluate the operator** (does the function name exist?)
2. **Evaluate the operands** from left to right (simplify the arguments to a single value)
   a. You may need to do steps 1-3 recursively here! (see next slide)
3. **Apply the operator to the operands** (run the code in the body of the function)



Functions: Call expressions (2 of 2)

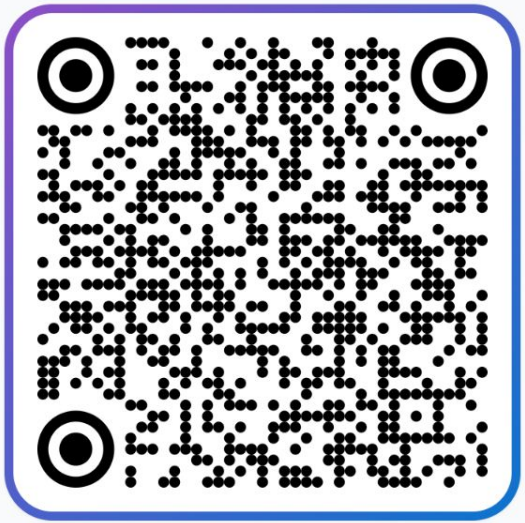


When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Functions: What would Python display?



Print vs. Return

- Use `print` if you want to display something
- Use `return` if you want to actually compute something
- Analogy: `print` is like if I have $5 and show you I have it. `return` is like I give you the $5 to use.
- Pure vs. non-pure functions
  - **Pure** functions only return a value, and given some input, it will always produce the same output
  - **Non-pure** functions produce side effects, such as changing or **mutating** some data structure, or printing

None

- What happens if you don't have a `return` statement?
  - Your function will implicitly return `None`, a special value in Python that represents "nothing"
  - `print` returns `None`
- **Note:** The value `None` is not displayed by the Python interpreter unless it is explicitly printed!

```
>>> def square(x):
...     x ** 2
>>> square(2)
>>> print(square(2))
None
```

Exceptions I

10 min

- Exceptions
- Reading a stack trace
- Print debugging
- Assert statements

Exceptions

- When something goes wrong in a program, an **exception** (or **error**) occurs
  - AKA "**raised**" or "**thrown**"
- Types of errors, in increasing order of difficulty of **debugging**
  - **Syntax**: "Illegal" Python code
  - **Runtime**: Python code is legal, but does something that breaks the program as it's running
  - **Logical**: Python code is legal and doesn't do anything that would break the program, but it produces output you didn't want

Exceptions: Syntax

```
>>> def square(x:
...     return x * x
...
  File "<python-input-21>", line 1
    def square(x:
               ^
SyntaxError: '(' was never closed
```

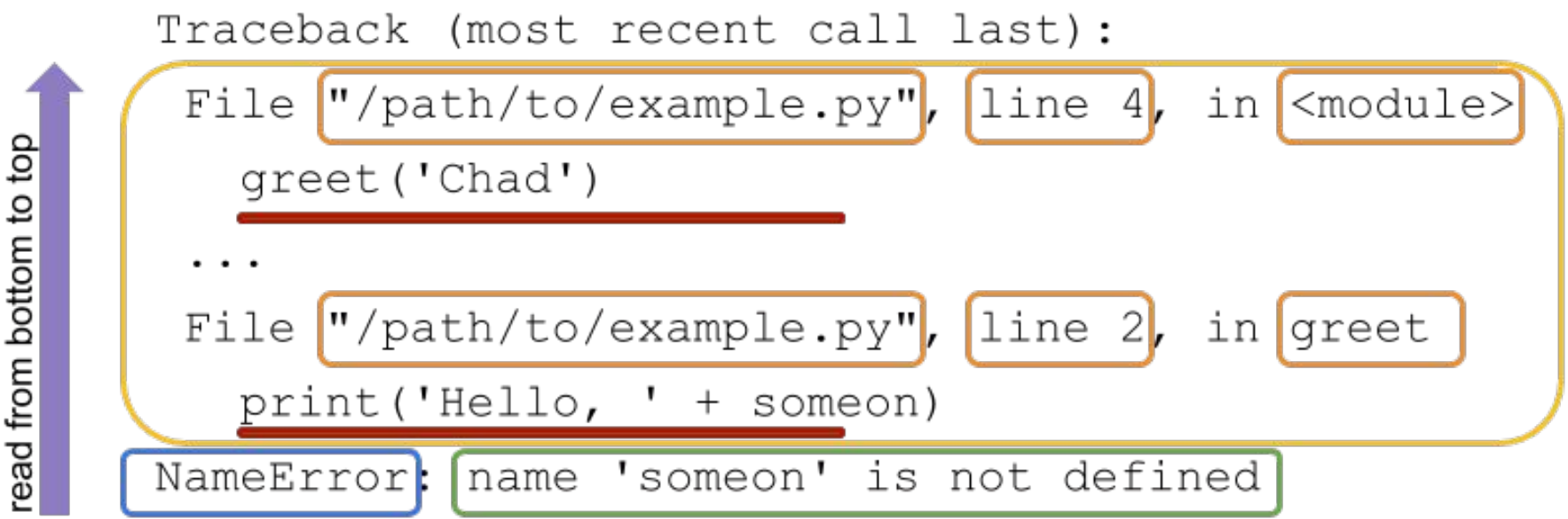
Exceptions: Runtime

```
>>> def divide(a, b):
...     return a / b
...
>>> divide(5, 0)
Traceback (most recent call last):
  File "<python-input-23>", line 1, in <module>
    divide(5, 0)
    ~~~~~~^^^^^^
  File "<python-input-22>", line 2, in divide
    return a / b
           ~~^~~
ZeroDivisionError: division by zero
```

Exceptions: Logical

```
>>> def square(x):
...     return x * 2
...
>>> square(3)
6
```

Reading a stack trace



Print debugging

- One debugging technique is **print debugging**
- This involves calling `print` on variables/values of interest throughout your program to determine where the program is going wrong
- In your assignments, you can do this by prefixing your print statements with `'DEBUG:'` so that the OkPy autograder doesn't autofail you

```
>>> from math import sqrt
>>> def quadratic_formula_positive(a, b, c):
...     discriminant = b * 2 - 4 * a * c
...     print("DEBUG: discriminant is", discriminant)
...     temp = (-b + sqrt(discriminant)) / (2 * a)
...     print("DEBUG: temp is", temp)
...     return temp
>>> quadratic_formula_positive(1, 2, 3)
```

Assert statements

Why debug when you can **prevent** bugs instead? → **Defensive programming**

One defensive programming technique is to use `assert` statements to programmatically verify your assumptions. If the assumption is `False`, an `AssertionError` is raised (with an optional message you provide).

```
>>> def divide(a, b):
...         assert b != 0, 'divisor cannot be 0'
...         return a / b
>>> divide(5, 0)
Traceback (most recent call last):
  File "<python-input-1>", line 1, in <module>
    divide(5, 0)
    ~~~~~~^^^^^^
  File "<python-input-0>", line 2, in divide
    assert b != 0, 'divisor cannot be 0'
           ^^^^^^
AssertionError: divisor cannot be 0
```