Computational Structures in Data Science

Lecture 2:
Abstraction and Functions

Computing In The News

COVID's long shadow looms over a new generation of college students

SFGate By Madilynne Medina, Jan 26, 2026

Emerging technologies like artificial intelligence and broader societal changes have created new challenges to learning and development. In Sharma's class, he noticed that over the past couple of years, students have moved away from using an online Q&A platform called "Ed Discussion" that allows students and professors to post questions. Now, students opt to use ChatGPT for help solving math problems, Sharma said, and he was surprised when students began "blatantly admitting" to using the AI tool.

"I don't say this to them, but it's like, of course you don't understand it because that's not the way we went over it," Sharma said. "ChatGPT doesn't understand your level. It just is searching through all math to find you an answer. That's not going to be a good way to learn it, especially because ChatGPT can't realize where you're struggling or figure out what is holding you up."

Announcements

- Join the EECS 101 and DATA 001 Ed Discussions!
  - https://eecs.link/join-ed
  - https://eecs.link/data-ed
- Hopefully not needed! *Please,* report any concerns about class / campus climate to the department, CS or DS. *You* are welcome here!
- https://eecs.link/climate

Announcements – Attendance & Sections

- https://edstem.org/us/courses/94478/discussion/7551104
- All sections happening this week.
- Please fill out the attendance form.

Links

- Q&A Thread: https://go.c88c.org/qa2
- Self-Check: https://go.c88c.org/2
- Website Google Calendar: https://c88c.org/fa23/weekly-schedule.html

Computational Structures in Data Science

Abstraction

Abstraction

- Detail removal

  "The act of leaving out of consideration one or more properties of a complex object so as to attend to others."

- Generalization

  "The process of formulating general concepts by abstracting common properties of instances"

- Technical terms: Compression, Quantization, Clustering, Unsupervized Learning

Henri Matisse *"Naked Blue IV"*

Experiment – **Where are you from?**

Where are you from?

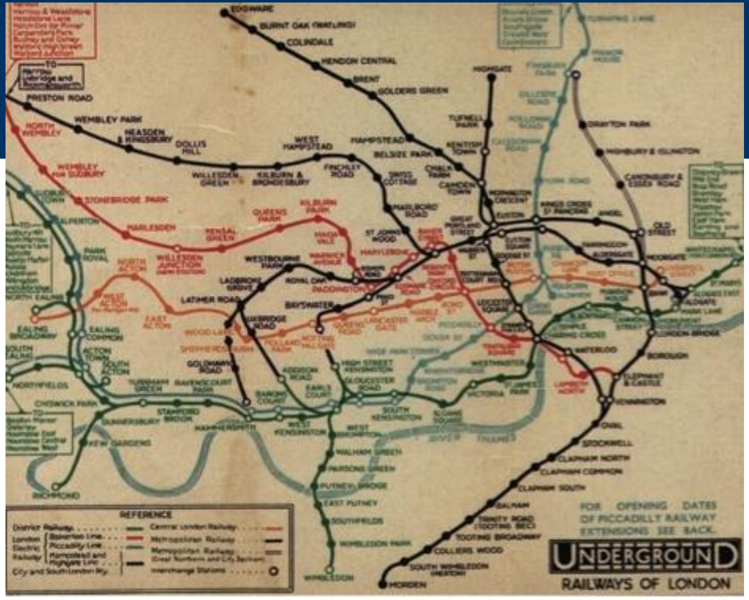
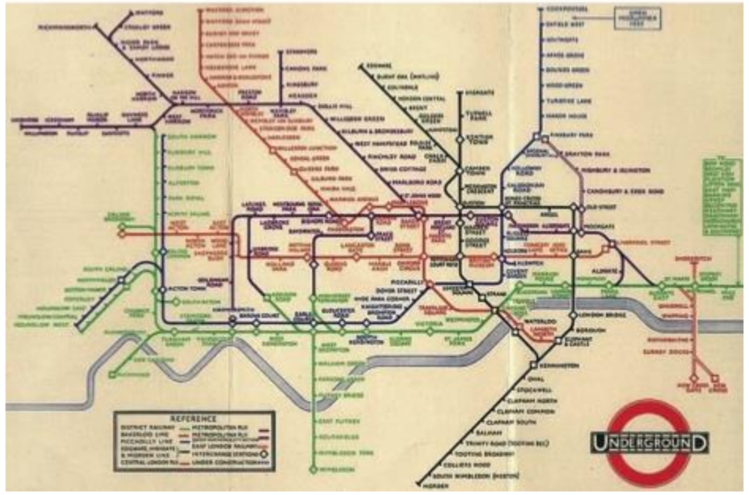
Possible Answers:

- Planet Earth
- Europe
- California
- The Bay Area
- San Mateo
- 1947 Center Street, Berkeley, CA
- 37.8693° N, 122.2696° W

All correct but different levels of abstraction!

Detail Removal (in Data Science)

- You'll want to look at only the interesting data, leave out the details, zoom in/out…
- Abstraction is the idea that you focus on the essence, the cleanest way to map the messy real world to one you can build
- Experts are often brought in to know what to remove and what to keep!



The London Underground 1928 Map & the 1933 map by Harry Beck.

The Power of Abstraction, Everywhere!

- Examples:
  - Math Functions (e.g., sin x)
  - Hiring contractors
  - Application Programming Interfaces (APIs)
  - Technology (e.g., cars)

- Amazing things are built when these layer
  - And the abstraction layers are getting deeper by the day!

We only need to worry about the interface, or specification, or contract NOT how (or by whom) it's built

Above the abstraction line

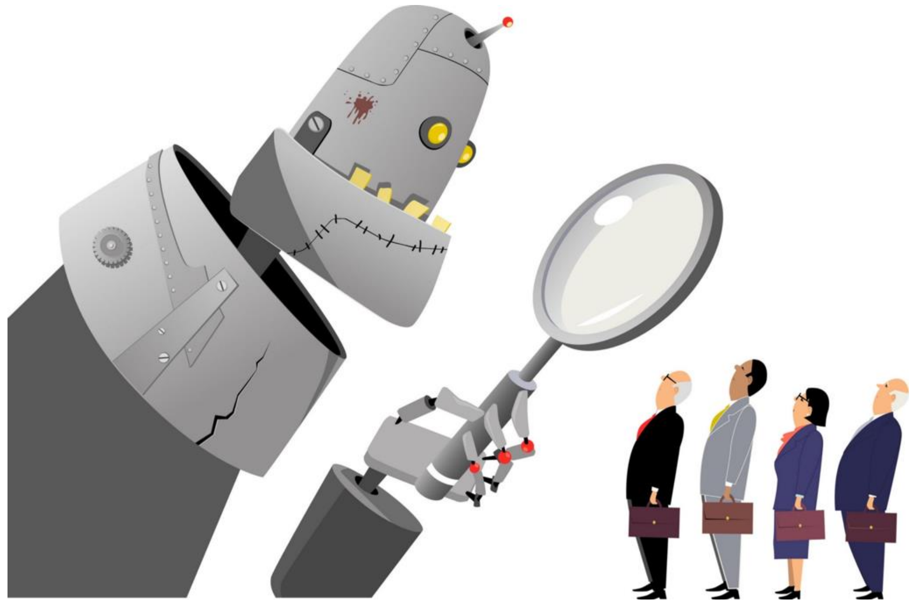
Abstraction Barrier (Interface)
(the interface, or specification, or contract)

Below the abstraction line

This is where / how / when / by whom it is actually built, which is done according to the interface, specification, or contract.

Abstraction: Pitfalls

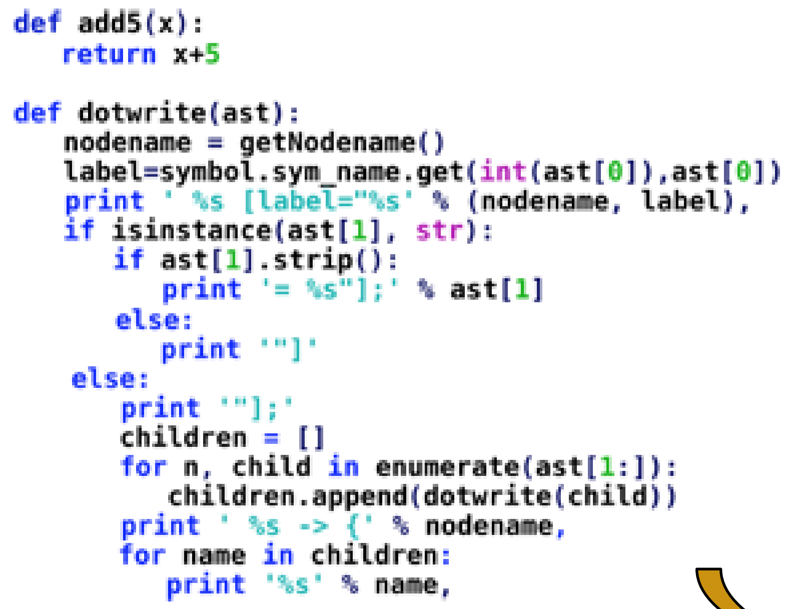
- Abstraction is not universal without loss of information (mathematically provable). This means, in the end, the complexity can only be “moved around”
- Abstraction makes us forget how things actually work and can therefore hide bias. Example: AI and hiring decisions.



- Abstractions can formalize a design or pattern. When something doesn't follow that pattern–perhaps a new use case emerges–it can be a burden to adapt.

Data or Code? Abstraction→ Take CS61C

Human-readable code (programming language)



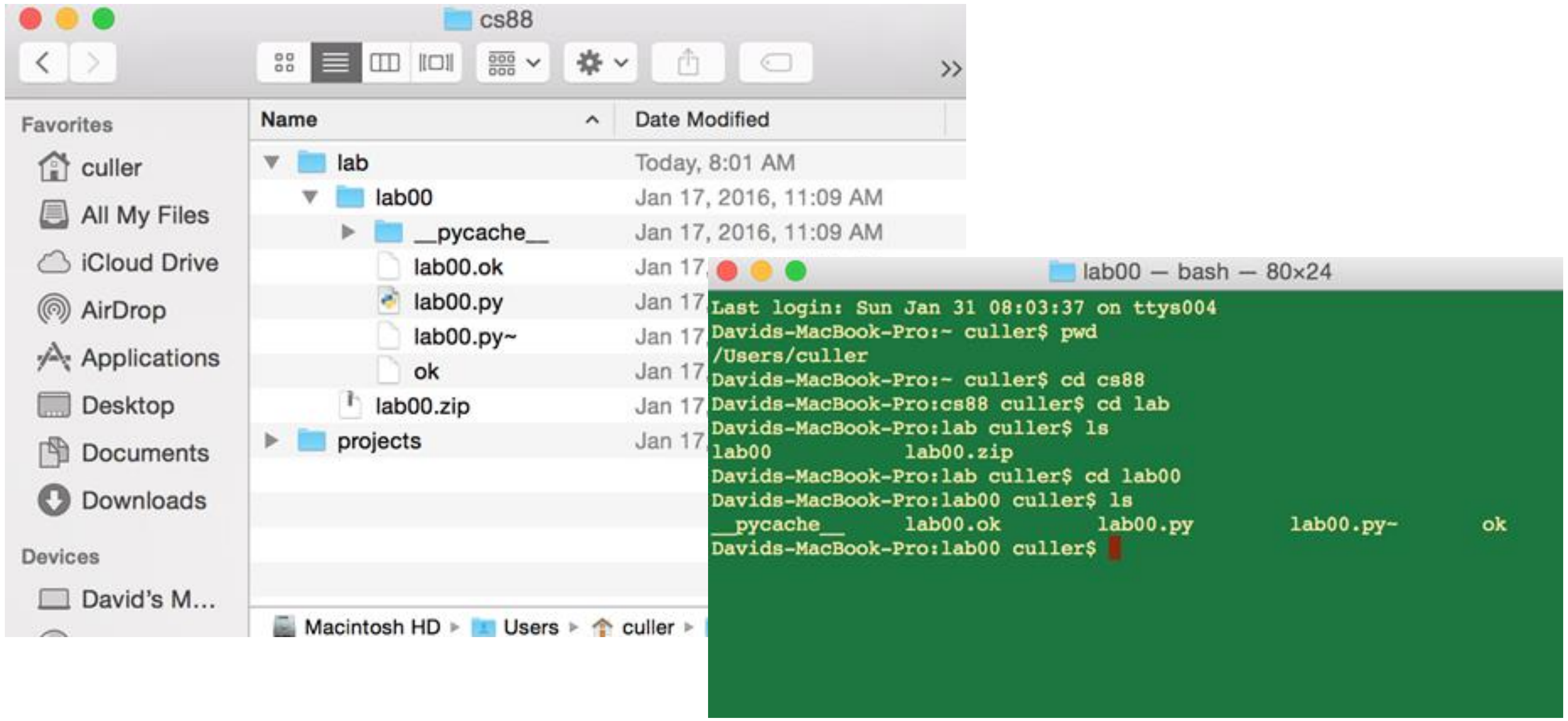
```
def add5(x):
   return x+5

def dotwrite(ast):
   nodename = getNodename()
   label=symbol.sym_name.get(int(ast[0]),ast[0])
   print ' %s [label="%s' % (nodename, label),
   if isinstance(ast[1], str):
      if ast[1].strip():
         print '= %s"];' % ast[1]
      else:
         print '"]'
    else:
      print '"];'
      children = []
      for n, child in enumerate(ast[1:]):
         children.append(dotwrite(child))
      print ' %s -> {' % nodename,
      for name in children:
         print '%s' % name,
```

Machine-executable instructions (byte code)

Compiler or Interpreter

Here: Python

Computers Are Built On Abstractions



Name	Date Modified
▼ lab	Today, 8:01 AM
▼ lab00	Jan 17, 2016, 11:09 AM
▶ __pycache__	Jan 17, 2016, 11:09 AM
lab00.ok	Jan 17
lab00.py	Jan 17
lab00.py~	Jan 17
ok	Jan 17
lab00.zip	Jan 17
▶ projects	Jan 17

```
Last login: Sun Jan 31 08:03:37 on ttys004
Davids-MacBook-Pro:~ culler$ pwd
/Users/culler
Davids-MacBook-Pro:~ culler$ cd cs88
Davids-MacBook-Pro:cs88 culler$ cd lab
Davids-MacBook-Pro:lab culler$ ls
lab00           lab00.zip
Davids-MacBook-Pro:lab culler$ cd lab00
Davids-MacBook-Pro:lab00 culler$ ls
__pycache__     lab00.ok        lab00.py        lab00.py~       ok
Davids-MacBook-Pro:lab00 culler$ 
```

- Big Idea: Layers of Abstraction
  - –The *GUI* look and feel is built out of files, directories, system code, etc.

Review:

- Abstraction:
  - Detail Removal or Generalizations
- Code:
  - Is an abstraction!

Computer Science is the study (and building) of abstractions

Python:
Expressions and Statements

Learning Objectives

- Evaluate expressions in Python
- *Name* data so it can be used later.
- Get practice with the Python Interpreter

Demo!

- Run the Python interpreter (`python3`) on your computer
- Practice seeing the results of expressions
- Use Control-L to clear the screen
- Use Control-D or type `exit()` to exit Python.
- The interpreter does not save any work!

Demo!

- Run the Python interpreter (`python3`) on your computer
- Practice seeing the results of expressions
- Use Control-L (`^L`) to clear the screen
- Use Control-D (`^D`) or type `exit()` to exit Python.

(^L)(^L)

- The interpreter does not save any work!

Python Interpreter Commands [Docs]

- `python3` – open a new interpreter with nothing loaded.
- `python3 [file.py]` – open a new interpreter and run/load file.py
- `python3 -i [file.py]` – after loading file.py stay in an *interactive session*
- Stylistic Notes:
  - `Monospace fonts` mean code or commands.
  - `[]` has many meanings.
    - In Python, this means a list (we'll ger to it)
    - In *documentation* this usually means an argument is *optional.*

Let’s talk Python

Expression	`max(88, 61)`
Call expression	`greeting`
Variables	`greeting = <expression>`
Assignment Statement	`def name(<arguments>):`
Define Statement	`if, else, for, while …`
Control Statements	`# Text are a # is ignored.`
Comments	

```
8 * 11
```

Expressions

An ***expression*** is code that produces or ***evaluates*** to a value.

A ***call expression*** simply means that expression involves calling a function.

```
8 * 11

8 + 80

max(88, 61)

len('Berkeley')
```

Names and Statements

- *Statements* are code that does something, but does not produce a value!
- *Assignment Statements* bind some value to a *name* which can be used later. (A *variable)*

```
print('Welcome to 88C!')
course = '88C'
print('Welcome to ' + course + '!')
```

Numbers (`int` and `float`)

- Numbers come in two types: integers, and decimals
  - Why? Partially historical reasons, partially for speed
- Python is forgiving!
  - In most cases you can mix them up just fine.
- Numbers support many common operations:
  - +, -, /, *, ** (power), % (modulus), // (floor division)
- Try: `import math`
- Lots of math examples

Strings and Text

- Data inside quotes "" is called a *string*
- Python allows single quotes or double quotes
- Strings support useful operations like concatenation with +
- "f-strings" allow us to nicely format text

- `f"Hello, {course}!"`
- `f"2 times 2 is {2*x}"`

Boolean Expressions

- Booleans are Yes/No values.
  - In Python: `True` and `False`
- `>, <, ==, !=, >=, <=, and, or`
  - Note the the "double equals"
- These expressions all return only True or False.
- `3 < 5` # returns True
- You can write `3 < 5 == True` – but this is redundant.
- We'll keep practicing over time

Boolean Expressions: and and or

- And and Or tell us the result of combining Boolean expressions
- and evaluates to `True` when both `a and b` are True
- `or` evaluates to `True` when either `a or b` is True

Expression	Result
`True and True`	`True`
`True and False`	`False`
`False and True`	`False`
`False and False`	`False`

Expression	Result
`True or True`	`True`
`True or False`	`True`
`False or True`	`True`
`False or False`	`False`

Statements and Expressions: Review

- Expressions evaluate to a result
- We can combine expressions for more complex problems
- We assign names to values using =

Live Coding Demo

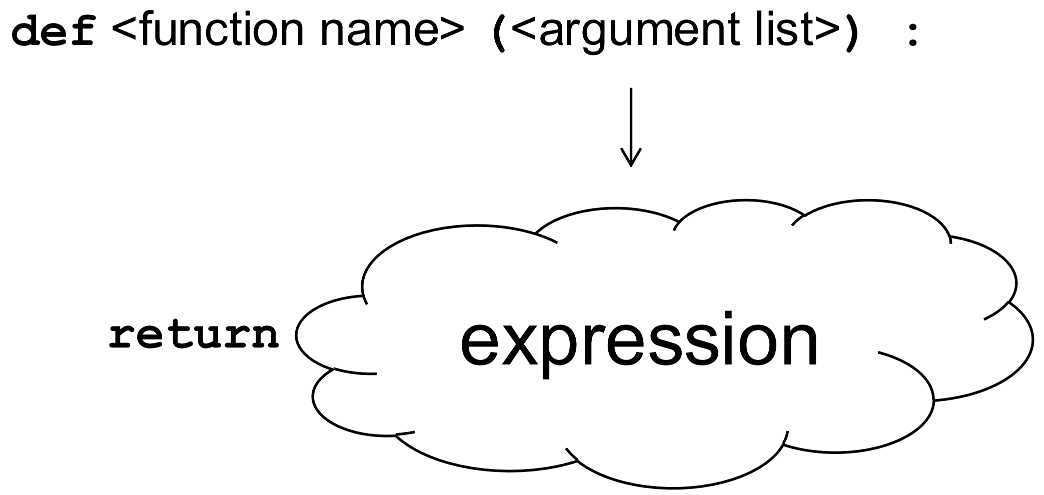
- Open Terminal on the Mac
- Type `python3`
  - We are now in the "interpreter" and can type code.
- Python runs each line of code as we type it.
  - After each line, we see a result. This happens *only* in the interpreter.
- It's a very useful calculator.
- We can also run files!
- `python3 -i 02-Functions.py`
  - `-i :` This means open the interpreter after running the file. It's optional
- `python3 ok …`
  - This runs the file "ok" which is included with each lab / homework.

Computational Structures in Data Science

Function Definitions

Defining Functions

- Abstracts an expression or set of statements to apply to lots of instances of the problem
- A function should do one thing well



Functions: Example

- Let's write a simple function which returns 8 more than the number.

- We will call this function by writing add_8(80).
- Inside, the name num will become the value 80.

```
def add_8(num):
    """add 8 to the input num
    >>> add_8(80)
    88
    """
    return 8 + num
```

Functions in Python

- We "define" them with def
- We typically name_them_using_underscores ("Snake case")
- The first line ends in a :
- The body is indented by 4 spaces (or 1 tab)
- Arguments (parameters) create 'names' that exist only in our function
- All functions return some value
  - We usually use `return`
  - If we omit return, the value is None

Function Arguments

- When we define a function, we provide 0 or more *arguments*
- Arguments define names that exist only within the function
- When we call a function, we pass *parameters* to the function
- Each parameter is mapped 1-to-1, left-to-right to an argument

```
def is_even(x):
     return x % 2 == 0

is_even(2)
```

Functions: Example

```
>>> y = 5
>>> x = 3
>>> z = max(x, y)
>>> z
5
```

```
def max(x, y):
    if x > y:
        return x
    else:
        return y
```

How to Write a Good Function

- Give a descriptive name
  - Function names should be lowercase. If necessary, separate words by underscores to improve readability. Names are extremely suggestive!
- Chose meaningful parameter names
  - Again, names are extremely suggestive.

Live Coding Demo

- Make and call simple functions

Computational Structures in Data Science

Functions and Environments

Functions: Calling and Returning Results

Python Tutor

```
def max(x, y):
    return x if x > y else y

x = 3
y = 4 + max(17, x + 6) * 0.1
z = x / y
```