

# Computational Structures in Data Science

---

## Lecture 2: Abstraction and Functions



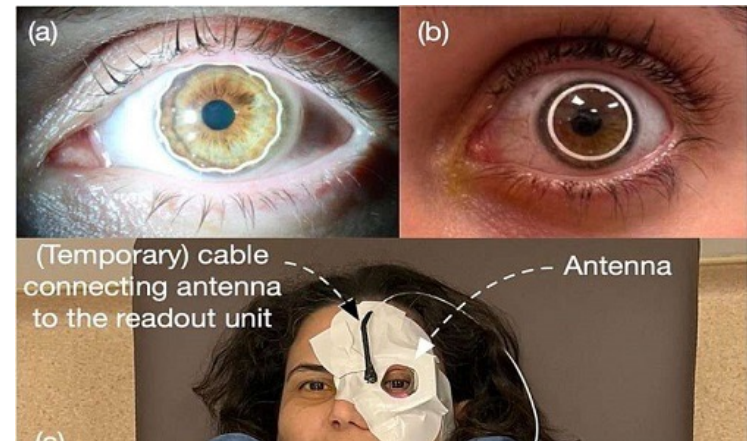
# Computing In The News

Photographs of subjects (a) wearing a Type-B contact lens

[Contact Lens Detects Signs of Glaucoma](#)

[The Engineer, Jan 17, 2024](#)

Contact lenses developed by researchers at the U.K.'s Northumbria University and Turkey's Bogaziçi University can detect glaucoma with embedded micro-sensors that track changes in intra-ocular pressure (IOP). The GlakoLens contacts can collect data over a 24-hour period and send it wirelessly to the wearer's ophthalmologist. Tracking IOP for longer periods could improve diagnostic accuracy.



# 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 – Waitlist and Exams

- We are working to expand the course.
  - Usually ~10% of enrollment gets off the waitlist (~50 students).
  - **Keep up with the class!**
- CalCentral:
  - You need to be in a section to to be enrolled.
  - Expanding the class will hopefully make this easier.
  - Please reach out to **advisors** about enrollment q's.
- Exams (reminder):
  - Midterm: Thurs March 14
  - Final: Tues May 7

## 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, Unsupervised Learning



Henri Matisse “Naked Blue IV”

# Experiment – Where are you from?

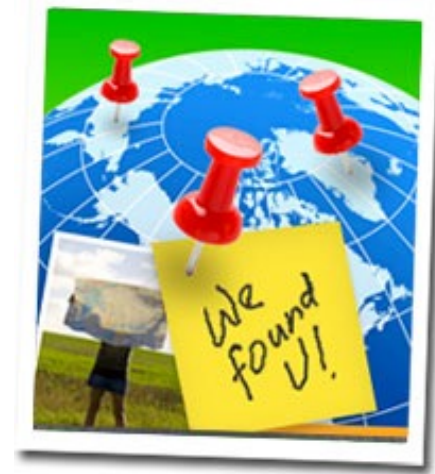




# Where are you from?

Possible Answers:

- Planet Earth
- Europe
- California
- The Bay Area
- San Mateo
- 1947 Center Street, Berkeley, CA
- $37.8693^{\circ}$  N,  $122.2696^{\circ}$  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)

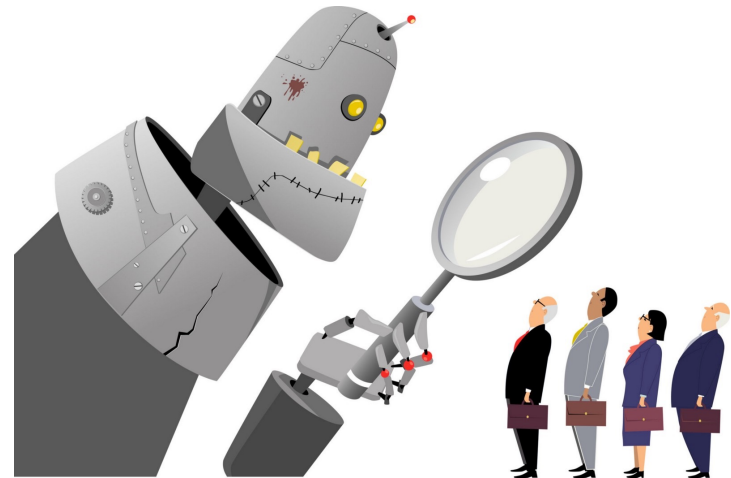
**Below the abstraction line**

*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)

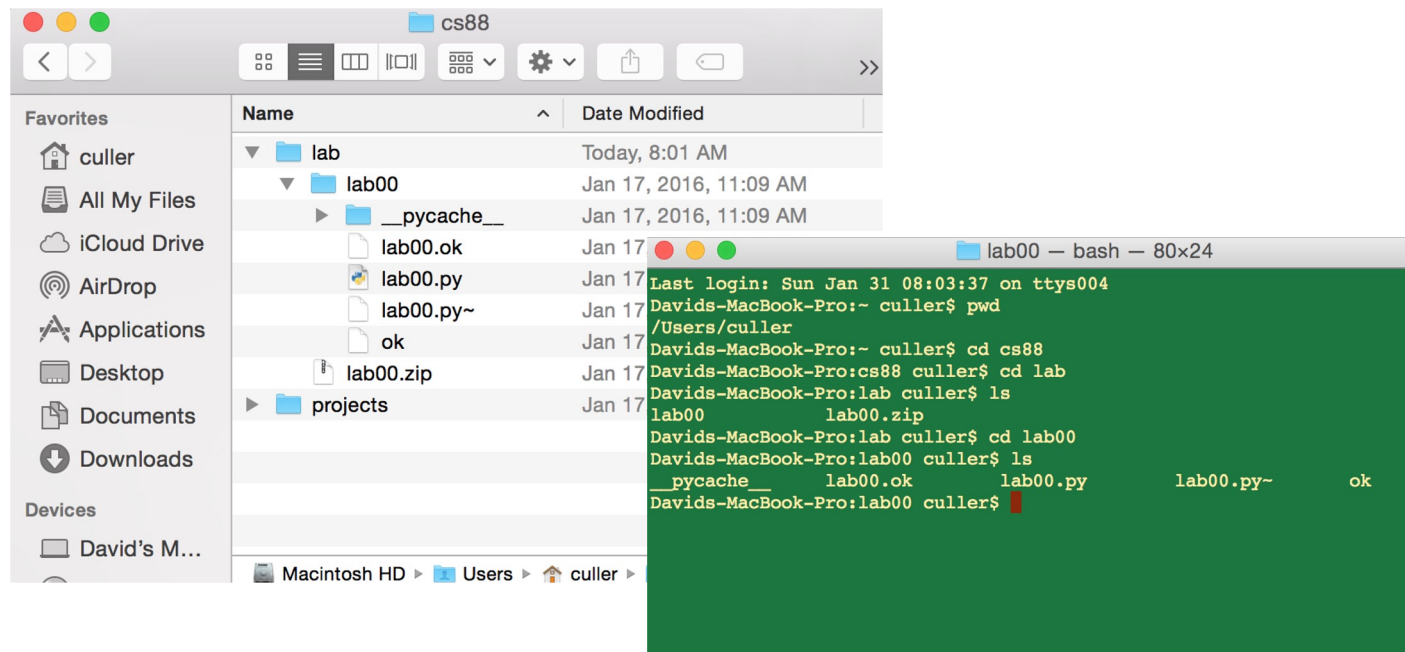
```
def add5(x):  
    return x+5  
  
def dotwrite(ast):  
    nodename = getNodeName()  
    label=symbol.sym_name.get(int(ast[0]),ast[0])  
    print ' %s [label="%s' % (nodename, label),  
    if isinstance(ast[1], str):  
        if ast[1].strip():  
            print '= %s";' % ast[1]  
        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)

```
0111001111000100110110000100011101000100111111011000111  
10111011000000111111011110011000011111111111110111110  
1111111111001111010001101010011000111000100101111001000  
11110001101010110011110110111001001111011111111111111  
11001111111001100000000011011111010010110011111101111  
111111110000011100011100011111001110000000110101111110  
0000111010011100100111110111110000111111001100110001011  
100111110000110001100110101111001111100010111010111111  
100100111111110011101111000111111000110111110001111110  
1101111011101011110111001111111001111111001111000100111  
11111000100101111000110001111100011111111111111110111  
11101111111100001110000010111100111111110000000111001100  
1010000011100111111011111111111100000000110001000011000  
11100111011011101111111110010111110111110000001111111  
1100110011000100001000111111110001111100100000100001000  
000011111101110010011100001111110111111111111111000100111  
1000011001100101110010001100010011011111000011000111111  
0011110011111100111110011110011101101111111110010111111  
11100111111101111000100111111110111111100111111110000  
01011011011101101111111101001101010101111111101000010
```

Compiler or Interpreter  
Here: Python

# Computers Are Built On Abstractions



- 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

# Computational Structures in Data Science

---

## 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!

# Let's talk Python

Expression

Call expression

Variables

Assignment Statement

Define Statement

Control Statements

Comments

```
max(88, 61)
```

```
greeting
```

```
greeting = <expression>
```

```
def name(<arguments>):
```

```
if, else, for, while ...
```

```
# Text are a # is  
ignored.
```

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

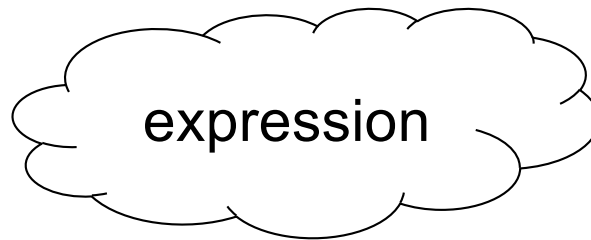
- Abstracts an expression or set of statements to apply to lots of instances of the problem
- A function should do one thing well

`def <function name> (<argument list>) :`



`return`

expression



# 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
```

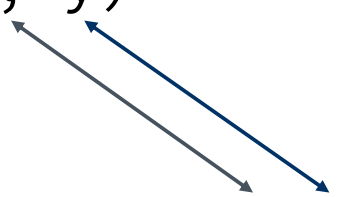
```
>>> x = 3
```

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

```
>>> z
```

```
5
```

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



The diagram consists of two arrows. A blue arrow originates from the 'y' parameter in the function definition 'def max(x, y):' and points to the 'y' argument in the function call 'max(x, y)' from the line '>>> z = max(x, y)'. A grey arrow originates from the 'x' parameter in the function definition and points to the 'x' argument in the same function call.

# 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
```