UC Berkeley EECS
Lecturer
Michael Ball

UC Berkeley EECS
Adj. Ass. Prof.
Dr. Gerald Friedland

Lecture 2: Abstraction and Functions

Announcements

- Sorry for the confusion in video posting…
  - Only need to watch 1 version.
- https://edstem.org/us/courses/3808/discussion/216190
  - Labs start Wednesday

UC Berkeley EECS
Lecturer
Michael Ball

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


Standard Time Zones of the World
WHERE ARE YOU FROM?
CANADA
U.S.
UNITED STATES
MEXICO
BRAZIL
ARGENTINA
CHILE
PERU
BOLIVIA
PARAGUAY
URUGUAY
VENEZUELA
COLOMBIA
ECUADOR
RUSSIA
MONGOLIA
CHINA
JAPAN
INDIA
PAKISTAN
KAZAKHSTAN
IRAN
IRAQ
SAUDI ARABIA
TURKEY
ALGERIA
LIBYA
EGYPT
SUDAN
NIGERIA
ETHIOPIA
KENYA
ZAIRE
ANGOLA
ZAMBIA
SOUTH AFRICA
MADAGASCAR
INDONESIA
PHILIPPINES
AUSTRALIA
NEW ZEALAND
North Atlantic Ocean
South Atlantic Ocean
North Pacific Ocean
South Pacific Ocean
Indian Ocean
Coordinated Universal Time (U.T.C.) formerly Greenwich Mean Time
International Date Line

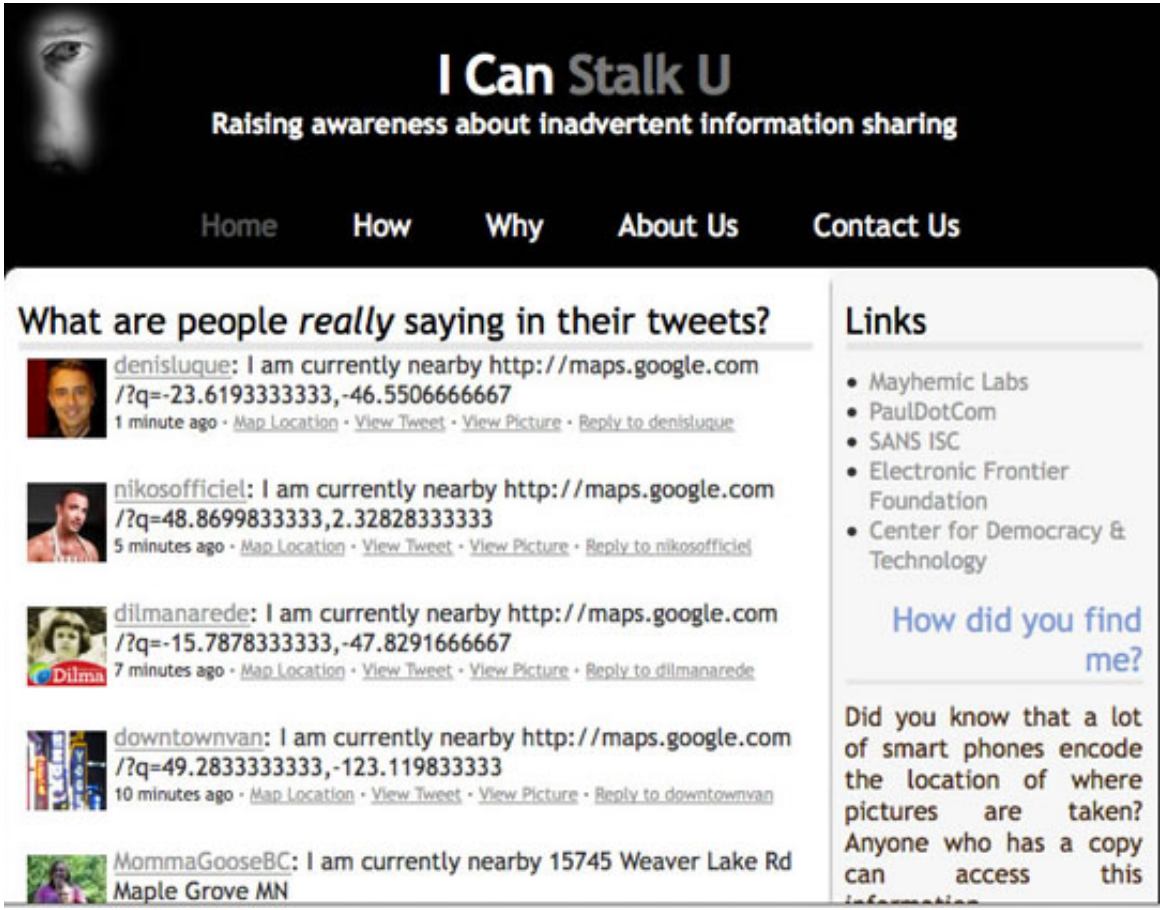
Where are you from?

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!

Abstraction gone wrong!



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:
  - 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)

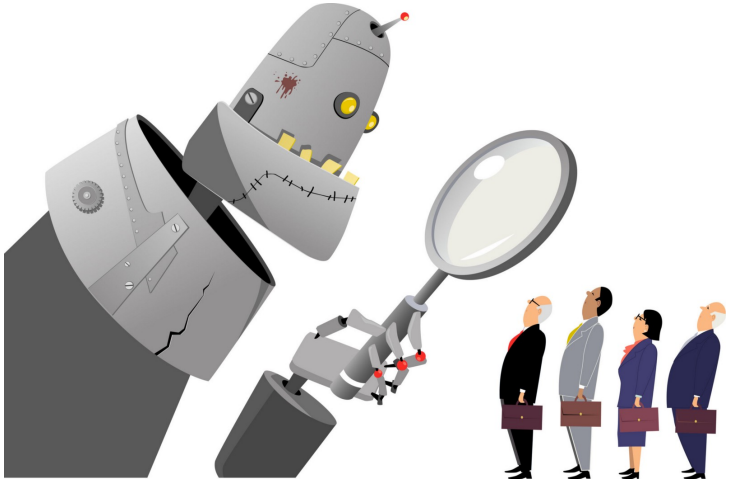
Below the abstraction line

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.

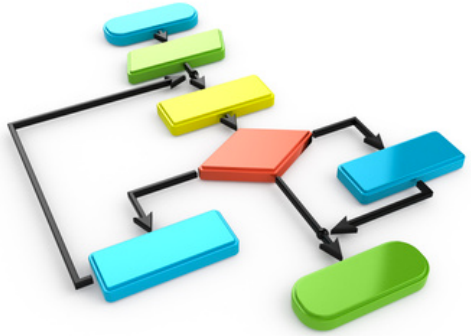


- Abstraction makes things special and that creates dependencies. Dependencies grow longer and longer over time and can become unmanageable.

Algorithm

- An algorithm (pronounced AL-go-rith-um) is a procedure or formula to solve a problem.
- An algorithm is a sequence of instructions to change the state of a system. For example: A computer's memory, your brain (math), or the ingredients to prepare food (cooking recipe).

  Think Data 8: Change or retrieve the content of a table.



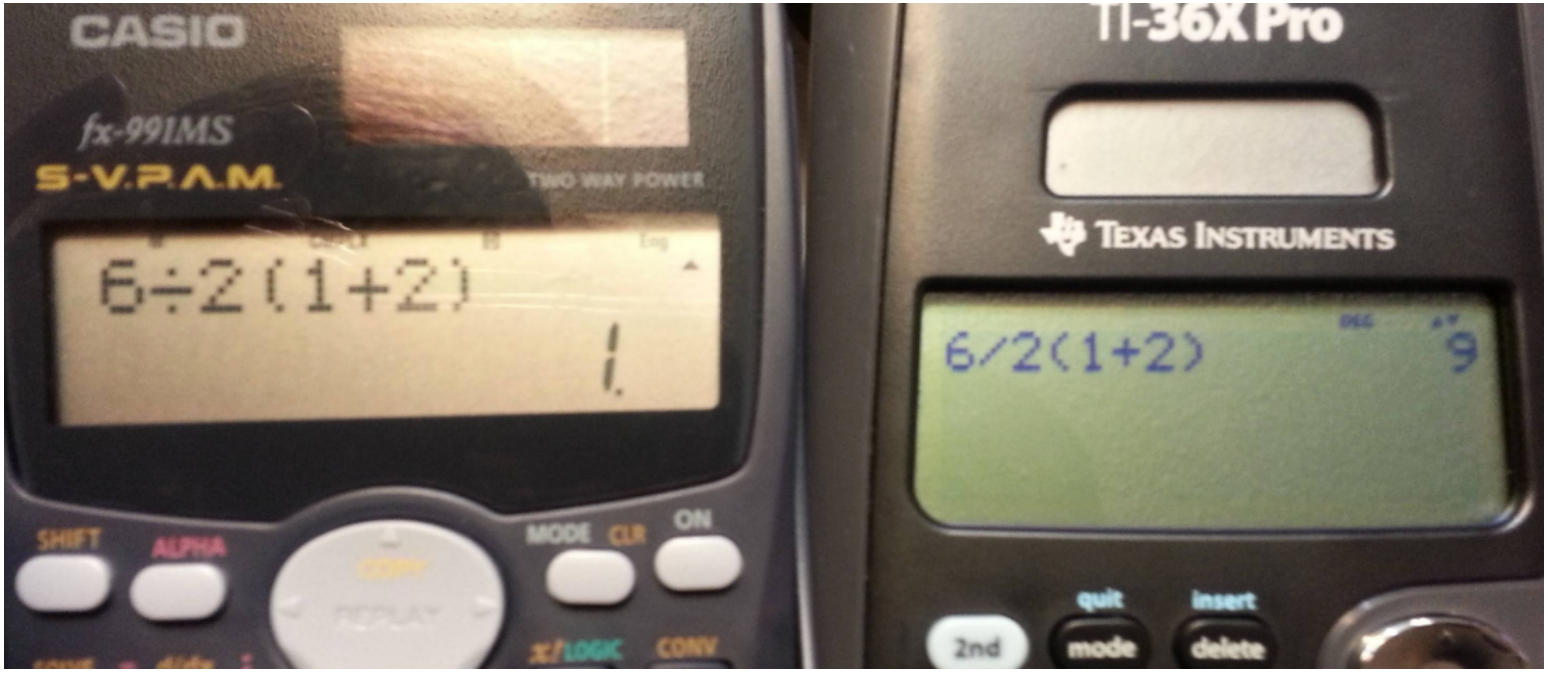
Algorithm: Properties

- An algorithm is a description that can be expressed within a finite amount of space and time.
- Executing the algorithm may take infinite space and/or time, e.g. ``calculate all prime numbers".
- In CS and math, we prefer to use well-defined formal languages for defining an algorithm.

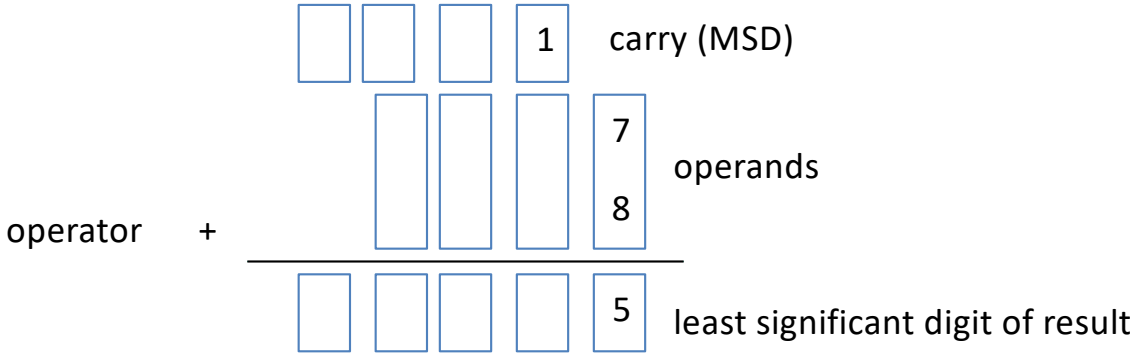
$$6 \div 2(1+2) = ?$$

1 or 9

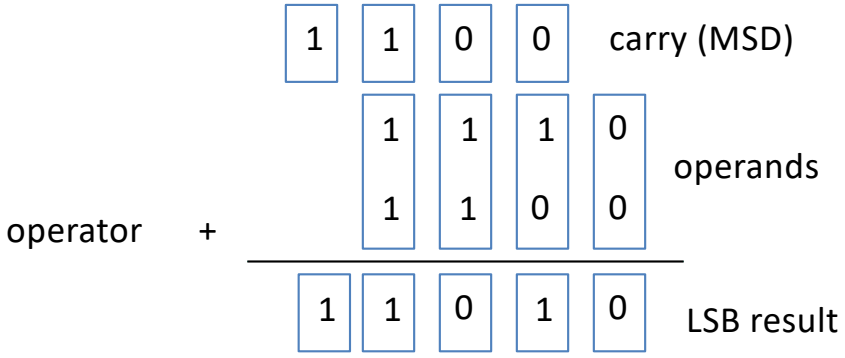
Algorithm: Well-Definition



Algorithms Early In Life (1st Grade)



Algorithms Early In Life (In Binary)



14
+ 12
26

More Terminology (Intuitive)

Code

A sequence of symbols used for communication between systems (brains, computers, brain-to-computer)

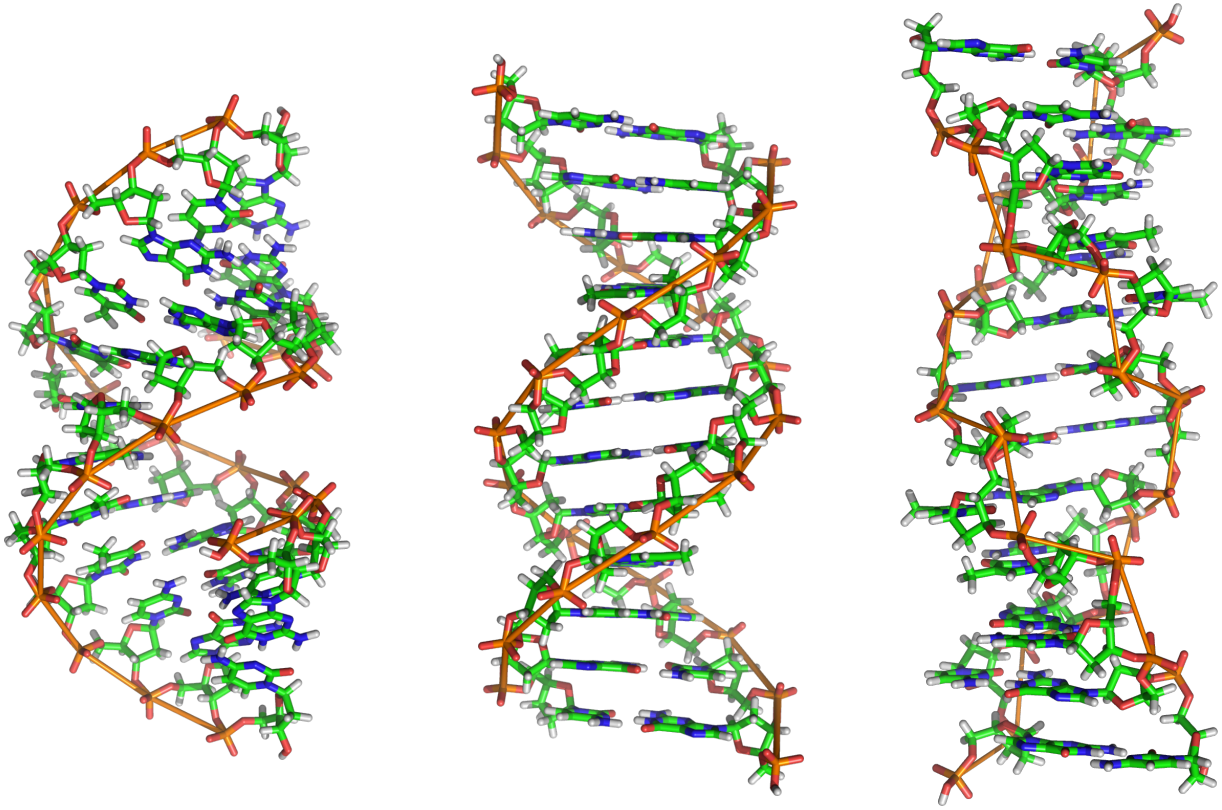
Data

Observations

Information

Reduction of uncertainty in a model (measured in bits)

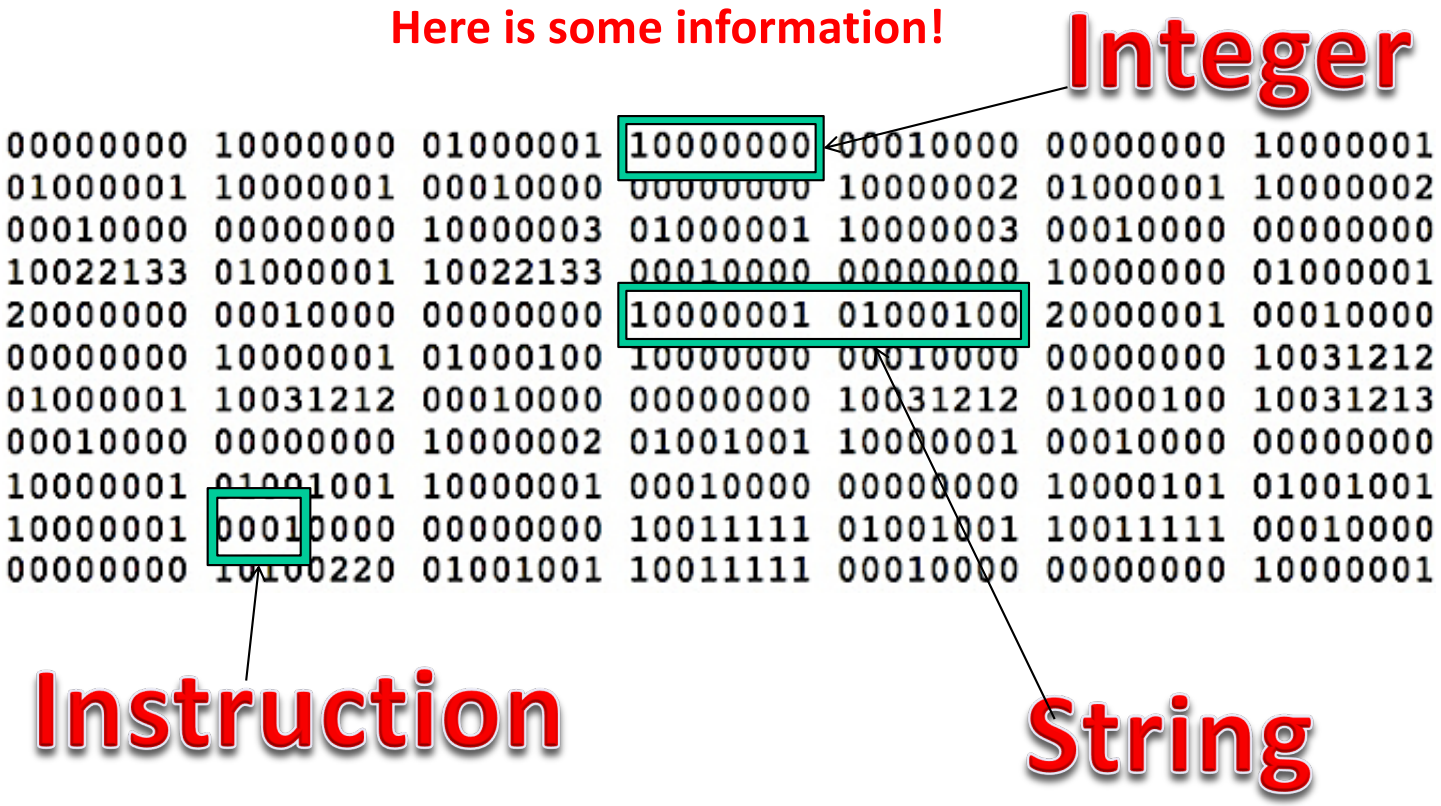
Data or Code?



Data or Code?

```
00000000 10000000 01000001 10000000 00010000 00000000 10000001
01000001 10000001 00010000 00000000 10000002 01000001 10000002
00010000 00000000 10000003 01000001 10000003 00010000 00000000
10022133 01000001 10022133 00010000 00000000 10000000 01000001
20000000 00010000 00000000 10000001 01000100 20000001 00010000
00000000 10000001 01000100 10000000 00010000 00000000 10031212
01000001 10031212 00010000 00000000 10031212 01000100 10031213
00010000 00000000 10000002 01001001 10000001 00010000 00000000
10000001 01001001 10000001 00010000 00000000 10000101 01001001
10000001 00010000 00000000 10011111 01001001 10011111 00010000
00000000 10100220 01001001 10011111 00010000 00000000 10000001
```

Data or Code?


Here is some information!
Integer
00000000 10000000 01000001 10000000 00010000 00000000 10000001
01000001 10000001 00010000 00000000 10000002 01000001 10000002
00010000 00000000 10000003 01000001 10000003 00010000 00000000
10022133 01000001 10022133 00010000 00000000 10000000 01000001
20000000 00010000 00000000 10000001 01000100 20000001 00010000
00000000 10000001 01000100 10000000 00010000 00000000 10031212
01000001 10031212 00010000 00000000 10031212 01000100 10031213
00010000 00000000 10000002 01001001 10000001 00010000 00000000
10000001 01001001 10000001 00010000 00000000 10000101 01001001
10000001 00010000 00000000 10011111 01001001 10011111 00010000
00000000 10100220 01001001 10011111 00010000 00000000 10000001
Instruction
String

Data or Code? Abstraction!

Human-readable code (programming language)

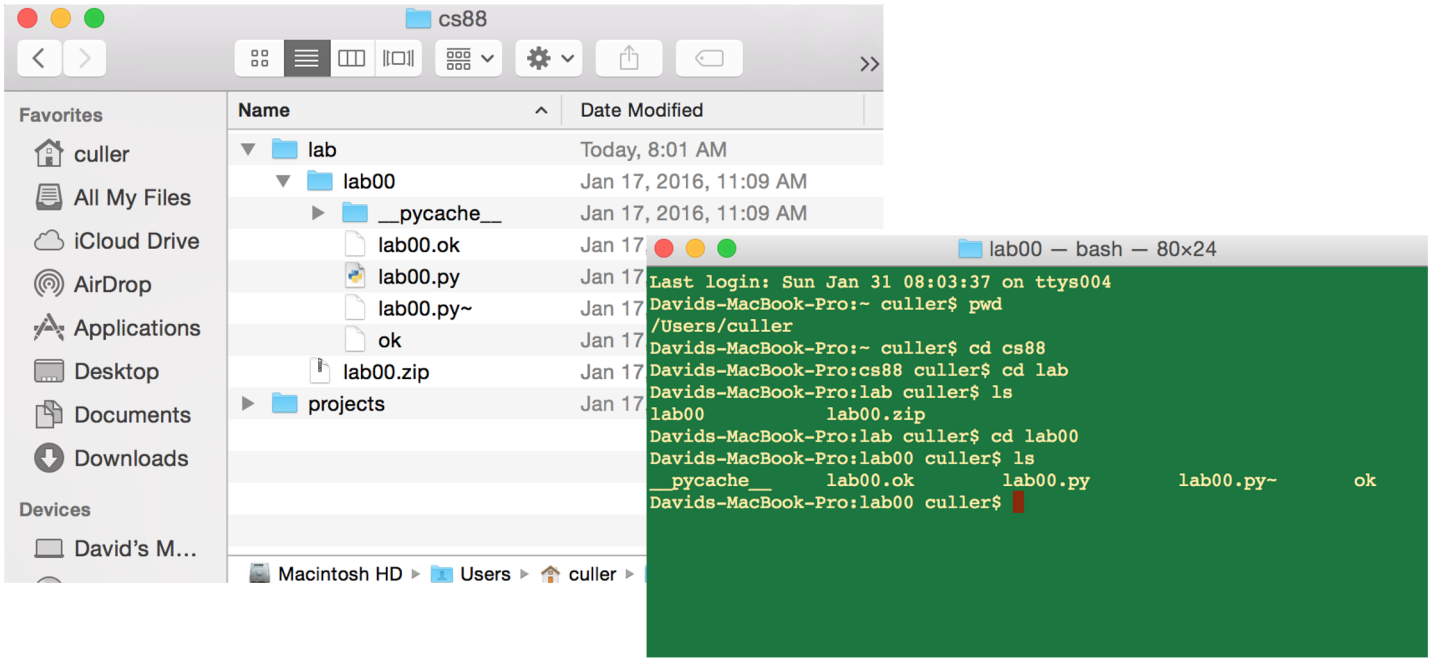
```
def add5(x):
   return x+5

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

Code or GUI: More Abstraction!



- 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!
  - Can be instructions or information

Computer Science is the study of abstraction

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Python: Statements and Functions

Learning Objectives

- Evaluate Python Expressions
- Call Functions in Python
- Assign data to Variables

Let's talk Python

- Expression `3.1 * 2.6`
- Call expression `max(0, x)`
- Variables `my_name`
- Assignment Statement `x = <expression>`
- Define Statement: **`def`** <function name> **`(`**<argument list>**`) :`**
- Control Statements:
  if …
  for …
  while …
  list comprehension

UC Berkeley EECS
Lecturer
Michael Ball

Python: Definitions and Control

Learning Objectives

- Create your own functions.
- Use if and else to control the flow of code.

Conditional Statement

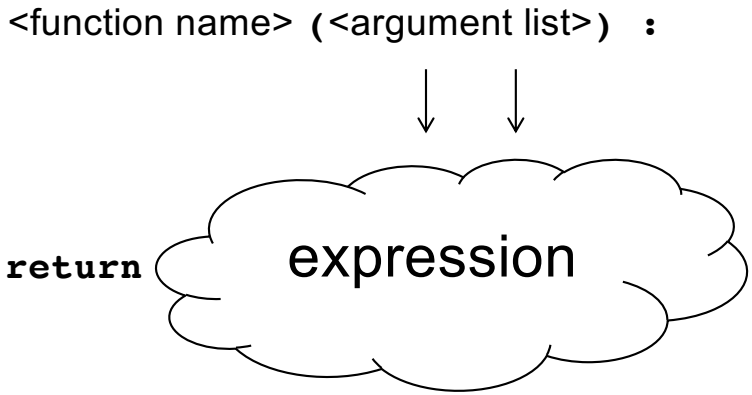
- Do some statements, conditional on a *predicate* expression

```
if <predicate>:
        <true statements>
else:
        <false statements>
```

- Example:

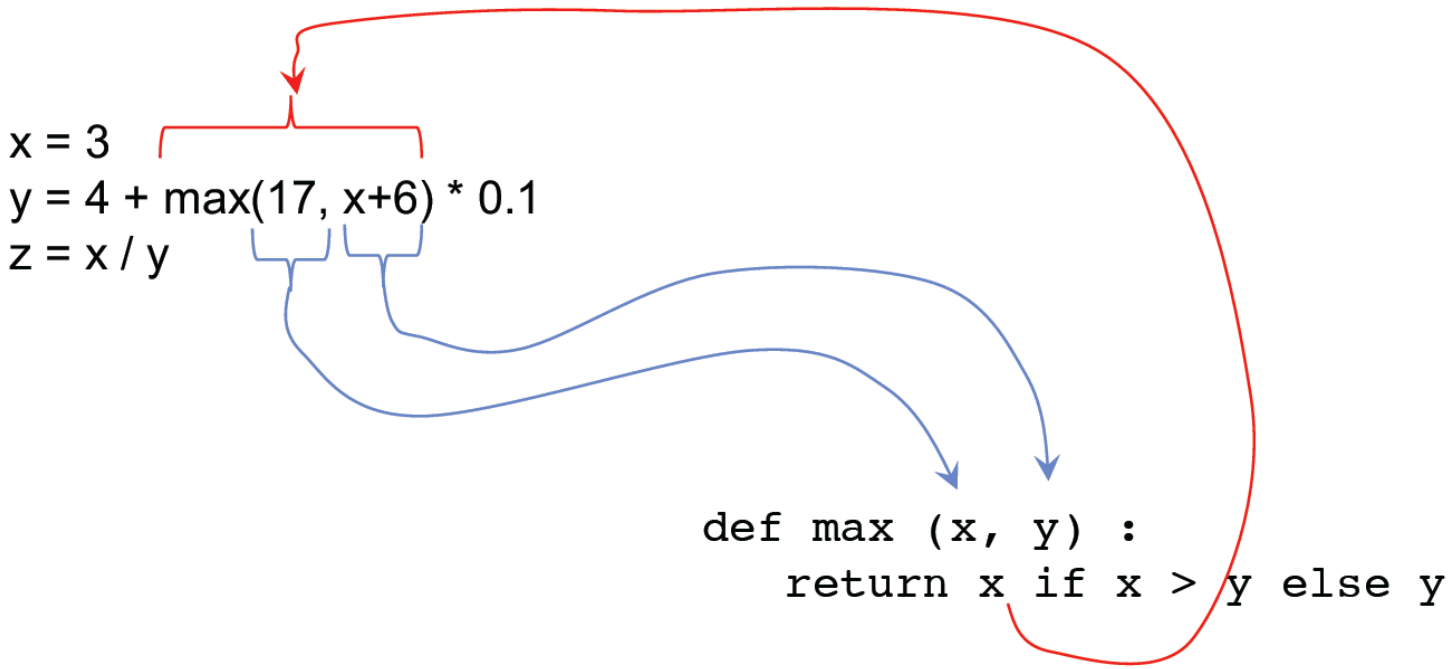
```
if (temperature>37.2):
        print("fever!")
else:
        print("no fever")
```

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



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.

- Write the docstring to explain *what* it does
  - What does the function return? What are corner cases for parameters?

Python Style Guide: https://www.python.org/dev/peps/pep-0008

- Write doctest to show what it should do
  - Before you write the implementation.

UC Berkeley EECS
Lecturer
Michael Ball

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
```

UC Berkeley EECS
Lecturer
Michael Ball

Iteration With While Loops

Learning Objectives

- Write functions that call functions
- Learn How to use while loops.

`while` *Statement – Iteration Control*

- Repeat a block of statements until a predicate expression is satisfied

```
<initialization statements>
while <predicate expression>:
    <body statements>

<rest of the program>
```