



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

Computational Structures in Data Science



Lecture #2: Algorithmic Structures



<https://www.wired.com/2016/09/heres-happens-two-designers-speak-infographics/>

January 26, 2018

<http://inst.eecs.berkeley.edu/~cs88>



Requirements for CS61b and CS Major

c8	CS88	CS47a	CS61b
----	------	-------	-------

CS major

c8	CS88	CS61b
----	------	-------

CS non-major, DS major

- **Data8+CS88 qualify you for CS61b**
- **Only CS majors:** Need to take CS47a **any time after CS88** to fulfill requirements.



Computational Concepts today

- Algorithm, Code, Data, Information
- Data Types, Simple Data Structures
- Function Definition Statement
- Conditional Statement
- Iteration



01/26/18

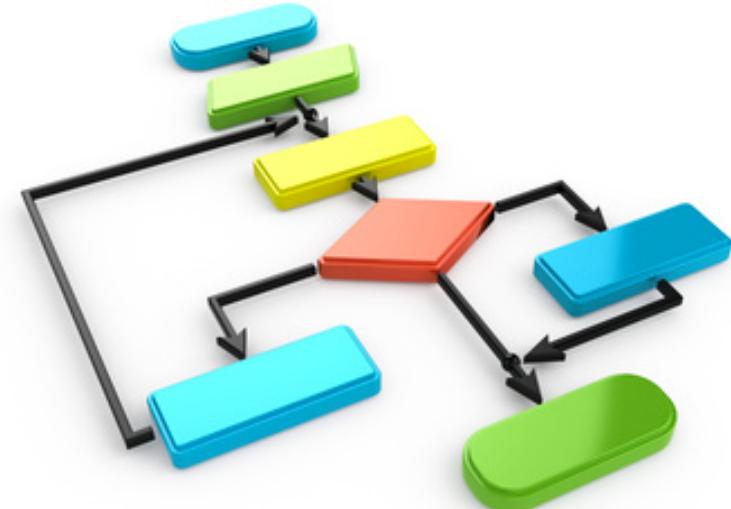
UCB CS88 Sp18 L2

3



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).





Algorithm: Properties

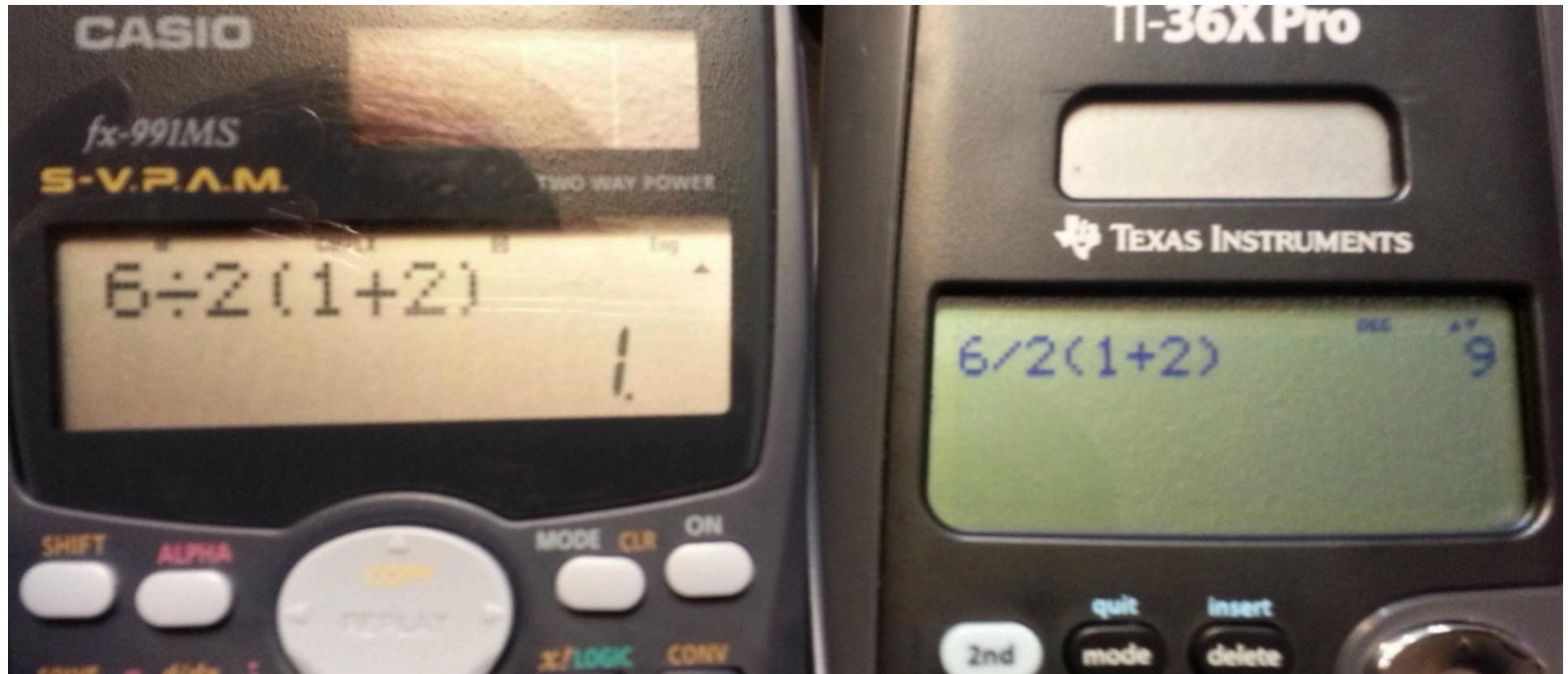
- An algorithm is a description that can be expressed within a finite amount of space and time.
- Executing the algorithm may take infinite space and/or time, e.g. “calculate all prime numbers”.
- In CS and math, we prefer to use well-defined formal languages for defining an algorithm.

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

1 or 9

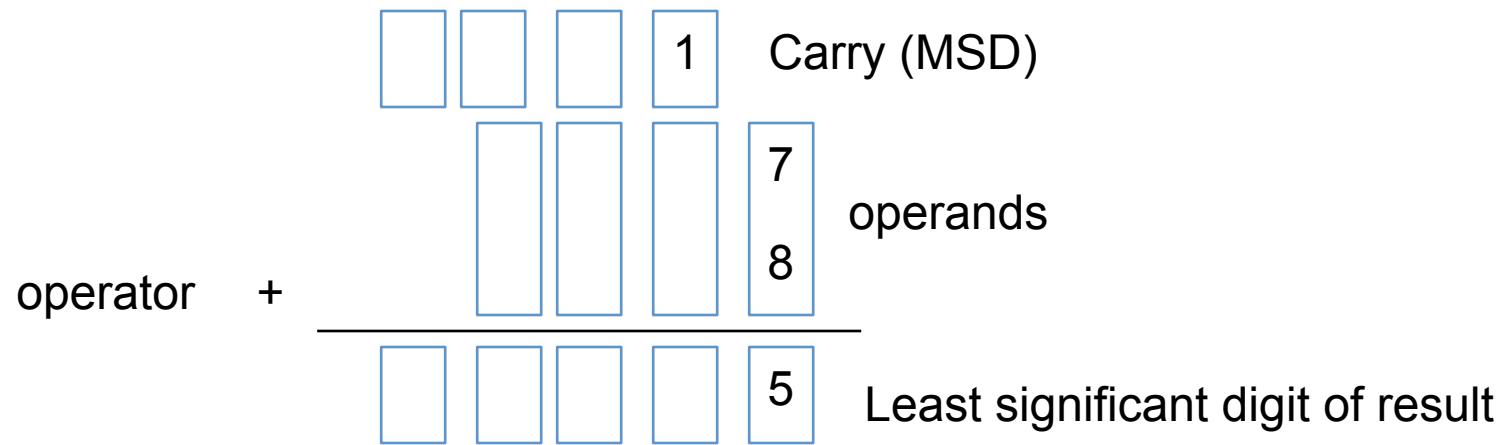


Algorithm: Well-definition





Algorithms early in life





Algorithms early in life (in binary)

operator +

1	1	0	0	Carry (MSD)
1	1	1	0	operands
1	1	0	0	
1	1	0	1	0
				LSB result

+ 14
+ 12
—
26



More Terminology (intuitive)

- **Code**

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

- **Data**

Collection of facts and inference (for reference or analysis)

- **Information**

Reduction of uncertainty about a fact (usually measured in bits)

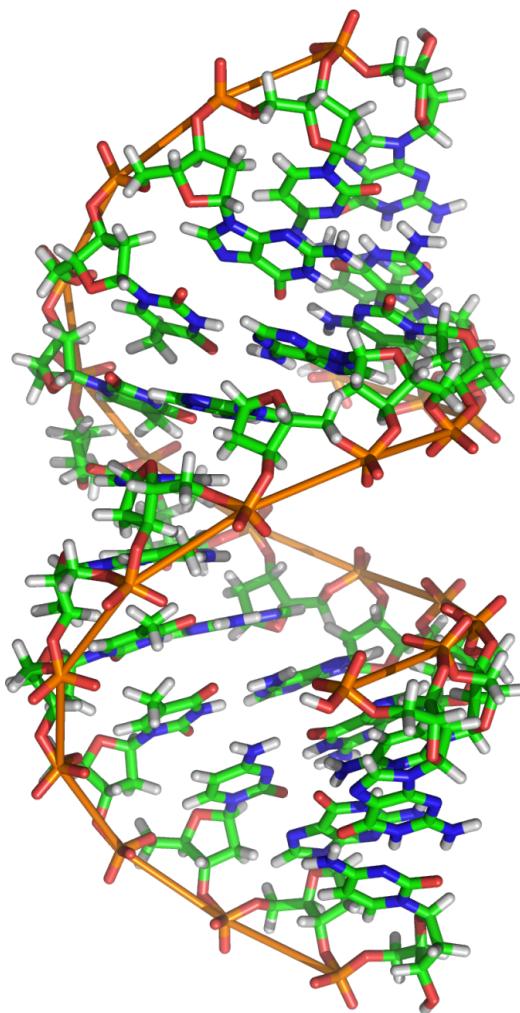


Experiment

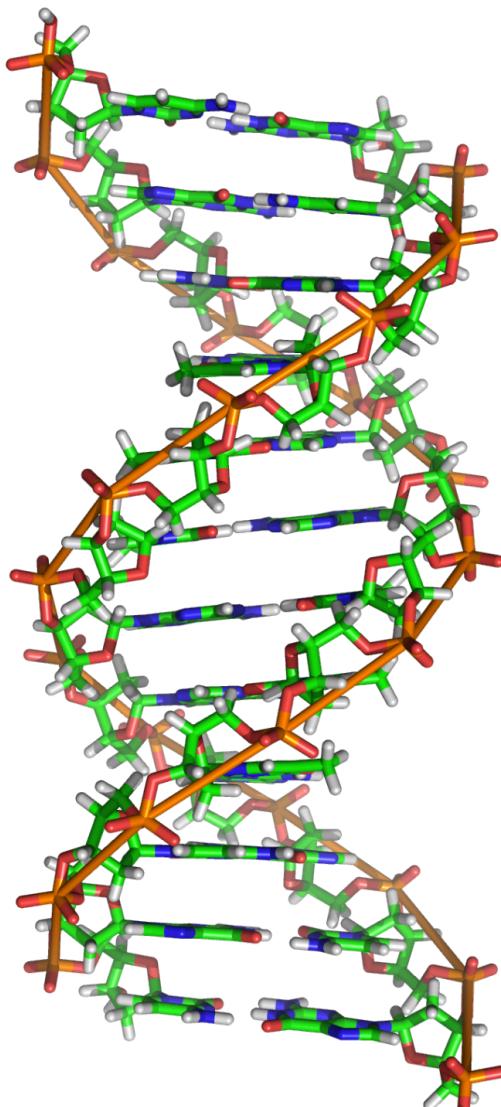
Code vs Data vs Information



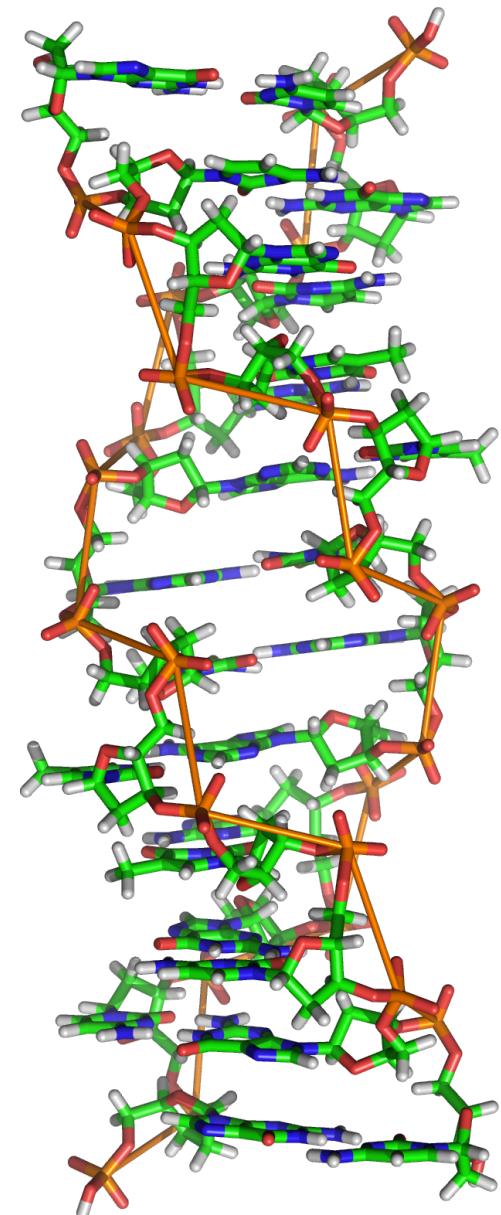
Data or Code?



01/26/18



UCB CS88 Sp18 L2



11



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!

00000000	10000000	01000001	10000000	00010000	00000000	10000001
01000001	10000001	00010000	00000000	10000002	00000000	10000002
00010000	00000000	10000003	01000001	10000003	00000000	10000000
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	1000220	01001001	10011111	00010000	00000000	10000001

Integer

Instruction

String



Data or Code?

**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 [%s]' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '=' % ast[1]
        else:
            print ''
    else:
        print "%s]" %
    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)**

```
01110011110001001101000010001110100010011111011000111
101110110000001111110111100110000111111111110111110
111111111100111010001101001100011100010010111001000
1111000110101011001111011011110010011110111111111111
1100111111100110000000000110111101001011001111110111
11111110000011100011100011111001111000000011010111110
00001110100111001001111011111000011111001100110001011
1001111100001100011001101111001111100010111010111111
1001001111111100111100011111100011011110001111110
110111101110101111011111001111111001111100100111
11111000100101111000110001111100011111111111110111
11101111111100001110000010111100111111100000000111001100
1010000011100111110111111111100000000110001000011000
11100111011011111100101111101111000000001111111111
11001100110001000010001111111100011111100100000100001000
000011111011001001110000111111011111111111000100111
100001100110010111001000100010011011111000011000111111
0011110011111100111110011111001101111111001011111110
111001111111011110001001111111011111110011111111110000
0101101101110101111111010011010101010111111101000010
```



**Compiler or Interpreter
Here: Python**



Language Structures (Python)

- **Variables** and **literals**
 - with some internal representation, e.g. Integers, Floats, Booleans, Strings, ...
In Python: Implicit data types!
- **Operations** on variable and literals of a **type**
 - e.g. +, *, -, /, %, //, **
 - ==, <, >, <=, >=
- **Expressions** are valid well-defined sets of operations on variables and literals that produce a value of a type.
 - `x=4*3`



More Language Structures (Python)

- **Data type: values, literals, operations**, e.g., int, float, string
- **Expression** $3.1 * 2.6$
- **Call expression** `max(0, x)`
- **Variables**
- **Assignment Statement** `x = <expression>`
- **Control Statement** `if ... (see later)`
- **Sequences: tuple, list** `(1, 2), [3, 4]`
 - `numpy.array(<object>)`
- **Data structures**
 - `numpy.array, Table`
- **Tuple assignment** `x, y = <expression>`



Call Expressions

- Evaluate a function on some arguments

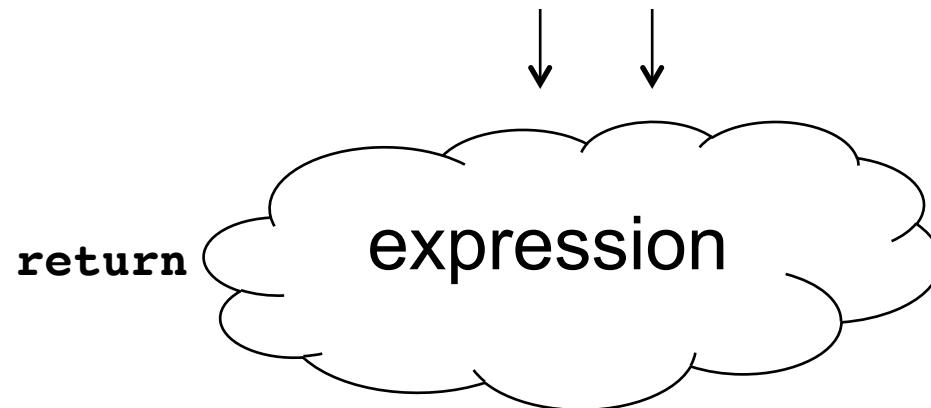
What would be some useful functions?

- Built-in functions
 - <https://docs.python.org/3/library/functions.html>
 - min, max, sum
- <https://docs.python.org/3/library/>
- str
- import math; help(math)



Defining Functions

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



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



Conditional statement

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

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



for statement – iteration control

- Repeat a block of statements for a structured sequence of variable bindings

```
<initialization statements>
for <variables> in <sequence expression>:
    <body statements>

<rest of the program>
```



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>
```



Data-driven iteration

- **describe an expression to perform on each item in a sequence**
- **let the data dictate the control**

```
[ <expr with loop var> for <loop var> in <sequence expr > ]
```



By the Way...

- Could we build a computer that has no instructions, only data?

Yes! The One Instruction Set Computer.

Check it out:

https://en.wikipedia.org/wiki/One_instruction_set_computer