Computational Structures in Data Science

Lecture #2: Algorithmic Structures

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

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**

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).**

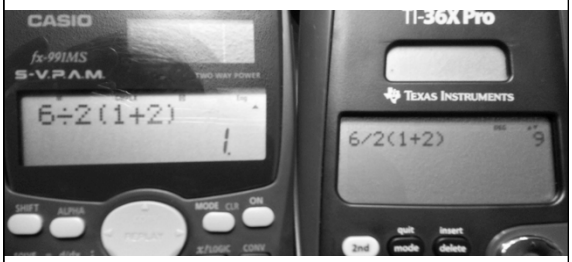
Algorithm: Properties

- **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.**

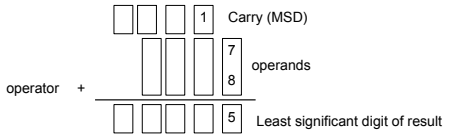
$$6 \div 2(1+2) = ?$$

$$1 \text{ or } 9$$

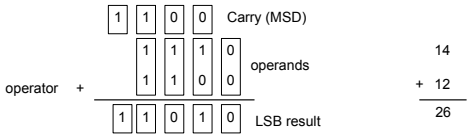
Algorithm: Well-definition



Algorithms early in life



Algorithms early in life (in binary)



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)

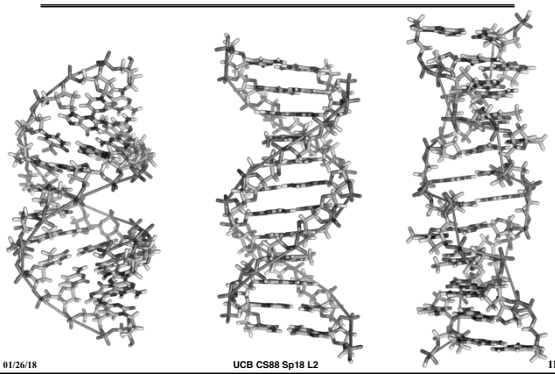
- **Information**

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

Experiment

Code vs Data vs Information

Data or Code?



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 [illegible] 00000002
00010000 00000000 10000003 01000001 10000003 [illegible] 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 10000220 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 [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

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>**) :**

return expression

- **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