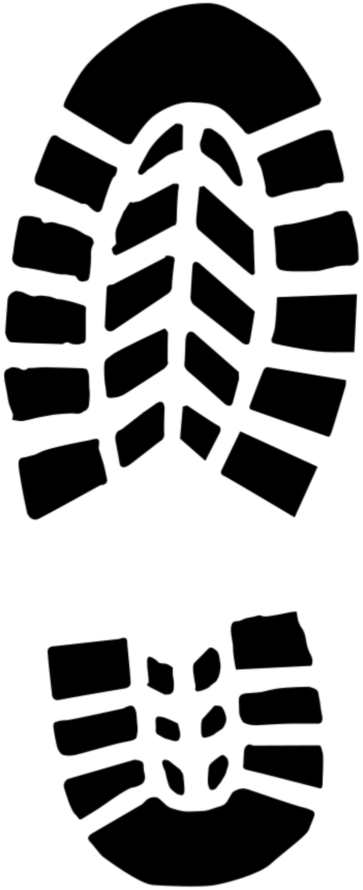


Welcome to

Python 2

Session #1

Michael Purcaro & The GSBS Bootstrappers

February 2014

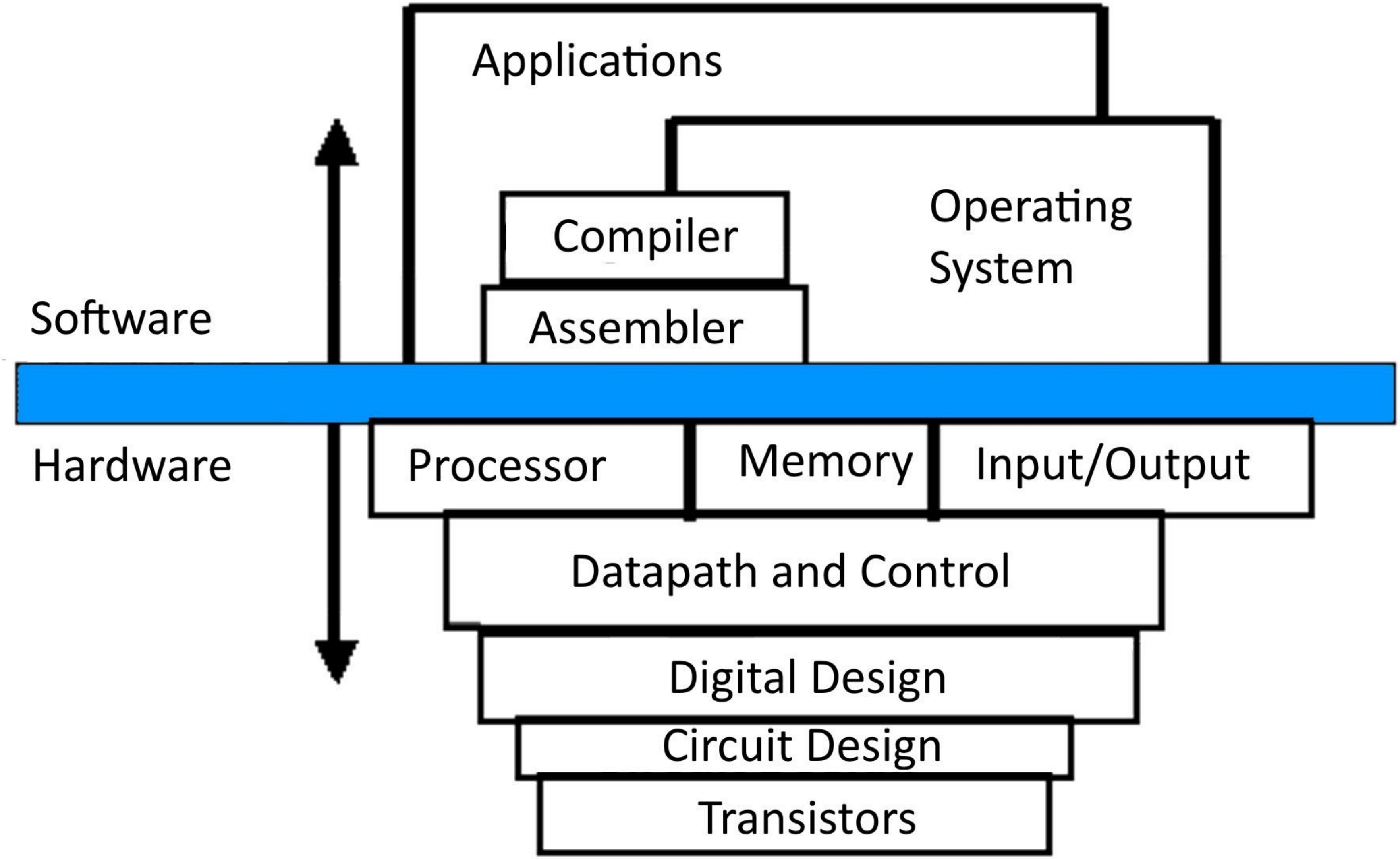
michael.purcaro@umassmed.edu

Welcome and Structure

- 6 sessions
  - Review of Python 1
  - Object orientation
  - Modules
  - Data structures
  - Regular Expressions
  - I/O
  - Working on cluster
- Work w/ real data

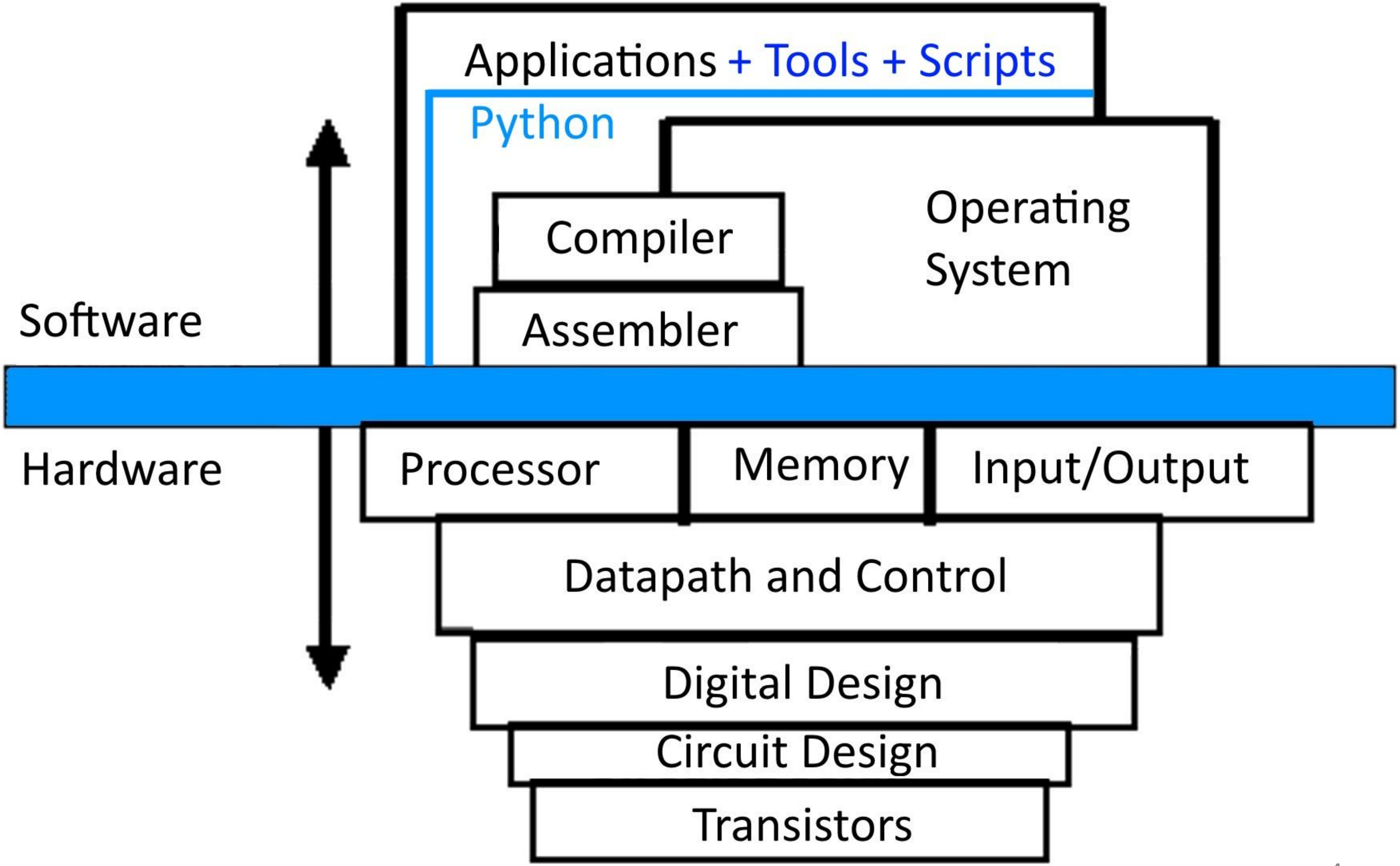
bioinfo.umassmed.edu/bootstrappers/bootstrappers-courses/python2/lecture1/

Layers of Abstraction



Based on http://www.cise.ufl.edu/~mssz/CompOrg/Figure1.1-LevelsOfAbstractn.gif

Layers of Abstraction



Computer Memory

Computer Memory: Addressing

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
…									

Computer Memory: Addressing

0	32	64	96	128	160	192	224	256	288
320	352	384	….						

units = bits

“data structure alignment”

Computer Memory: Addressing

0	32	64	96	128	160	192	224	256	288
320	352	384	….						

Memory size limited by system, number of programs running, etc.

Processors prefer to operate in terms of these blocks of memory

Example: adding two 3 digit intgers

$$\begin{array}{rrr} & & 6\\ + & & 7\\ \hline & 1 & 3\end{array}$$

Processors prefer to operate in terms of these blocks of memory

Base 10

	0	0	6
+	0	0	7
	0	1	3

Processors prefer to operate in terms of these blocks of memory

Base 10

+

0	0	6
0	0	7
0	1	3

Base 2 → 8 bit integers

0	0	0	0	0	1	1	0
0	0	0	0	0	1	1	1
0	0	0	0	1	1	0	1

Overflow

	1	1	1	1	1	1	1	1
+	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0

Result is too large to fit in memory given!

Building Blocks: Numbers

- Numbers take up a certain amount of space in memory
- Two fundamental types in python
  - Integers (ints)
  - Decimal (float)
- `print type(1)`
- `print type(1.2)`

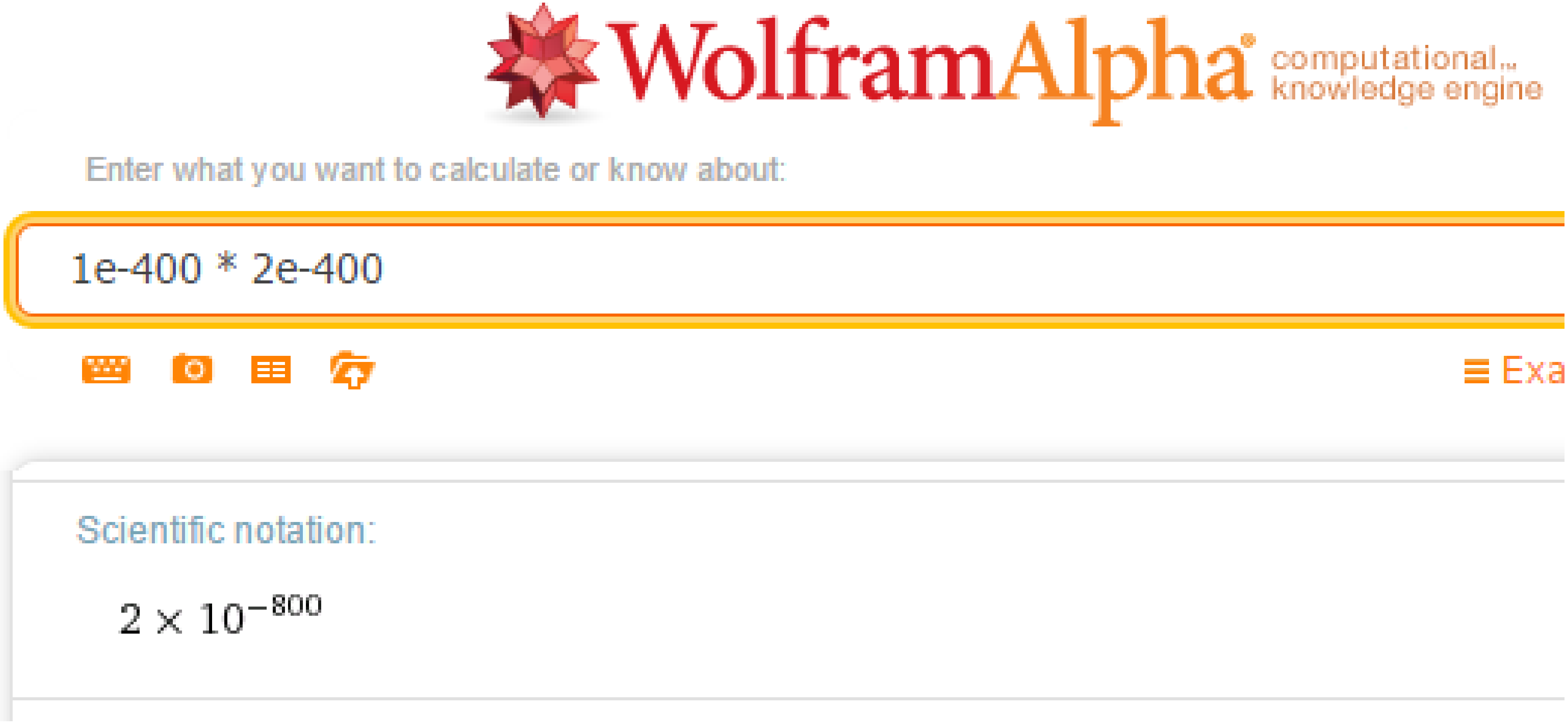
Size in memory limits precision of number types

```
print "1+2 = ", 1+2

print "1e100 + 2e100 = ", 1e100 + 2e100
print "1e400 + 2e400 = ", 1e400 + 2e400

print "1e-100 + 2e-100 = ", 1e-100 + 2e-100
print "1e-400 + 2e-400 = ", 1e-400 + 2e-400
```

Arbitrary precision numbers possible!



Enter what you want to calculate or know about:

```
1e-400 * 2e-400
```

Scientific notation:

2×10^{-800}

Arbitrary precision numbers in Python

```
import gmpy
from gmpy import mpf

print "mpf('1e100') + mpf('2e100') = ", mpf('1e100') + mpf('2e100')

print "mpf('1e400') + mpf('2e400') = ", mpf('1e400') + mpf('2e400')

print "mpf('1e-100') + mpf('2e-100') = ", mpf('1e-100') + mpf('2e-100')

print "mpf('1e-400') + mpf('2e-400') = ", mpf('1e-400') + mpf('2e-400')
```

Arbitrary precision numbers in Python

```
import gmpy
from gmpy import mpf
```

```
print "mpf('1e100') + mpf('2e100') = ", mpf('1e100') + mpf('2e100')
mpf('1e100') + mpf('2e100') =  3.e100

print "mpf('1e400') + mpf('2e400') = ", mpf('1e400') + mpf('2e400')
mpf('1e400') + mpf('2e400') =  3.e400

print "mpf('1e-100') + mpf('2e-100') = ", mpf('1e-100') + mpf('2e-100')
mpf('1e-100') + mpf('2e-100') =  3.e-100

print "mpf('1e-400') + mpf('2e-400') = ", mpf('1e-400') + mpf('2e-400')
mpf('1e-400') + mpf('2e-400') =  3.e-400
```

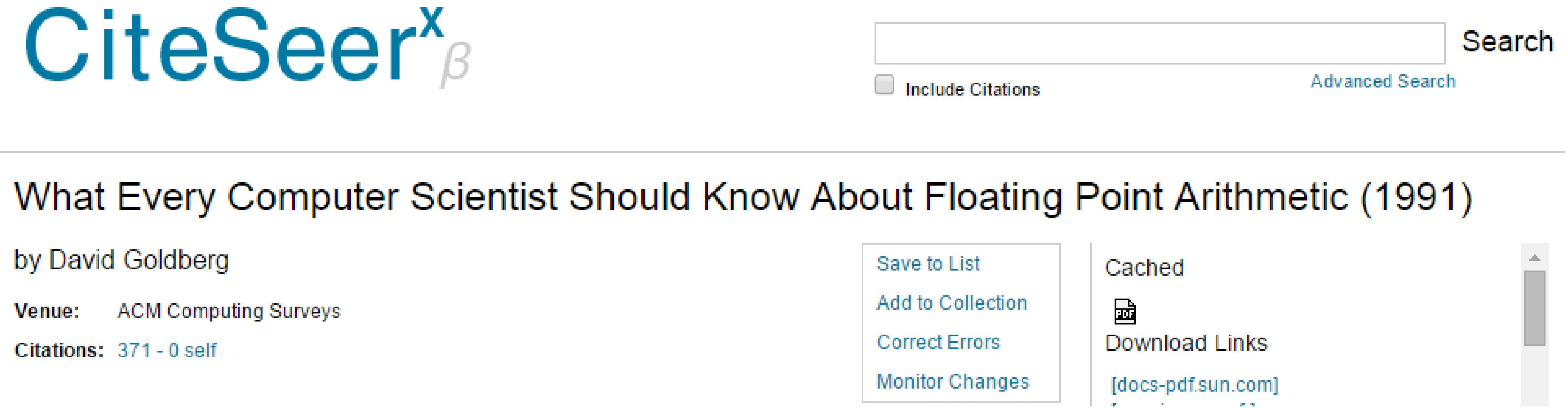
More examples in Lecture1/nums.py

Why aren’t all numbers arbitrary precision?

- Performance
- Size in memory
- Many complex and subtle floating point issues

Why aren’t all numbers arbitrary precision?

- Performance
- Size in memory
- Many complex and subtle floating point issues



CiteSeer^x β

Search

□ Include Citations

Advanced Search

What Every Computer Scientist Should Know About Floating Point Arithmetic (1991)

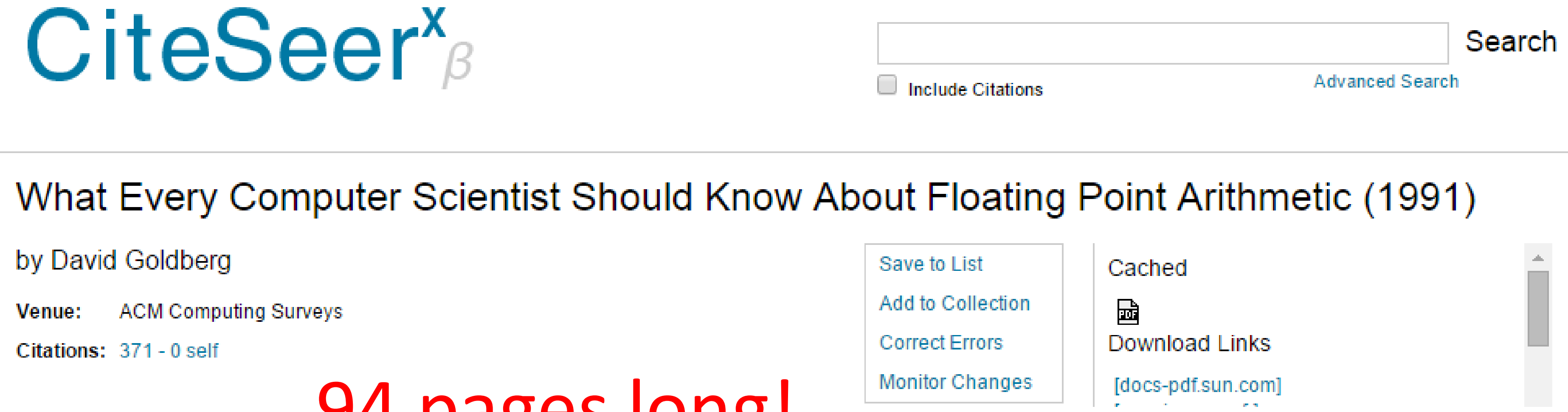
by David Goldberg

Venue: ACM Computing Surveys

Citations: 371 - 0 self

Why aren’t all numbers arbitrary precision?

- Performance
- Size in memory
- Many complex and subtle floating point issues



What Every Computer Scientist Should Know About Floating Point Arithmetic (1991)

by David Goldberg

Venue: ACM Computing Surveys

Citations: 371 - 0 self

94 pages long!

Building Blocks: Lists

- Contiguous set of blocks in memory

```
v = []
for i in range(10):
    v.append(i)
print v
```

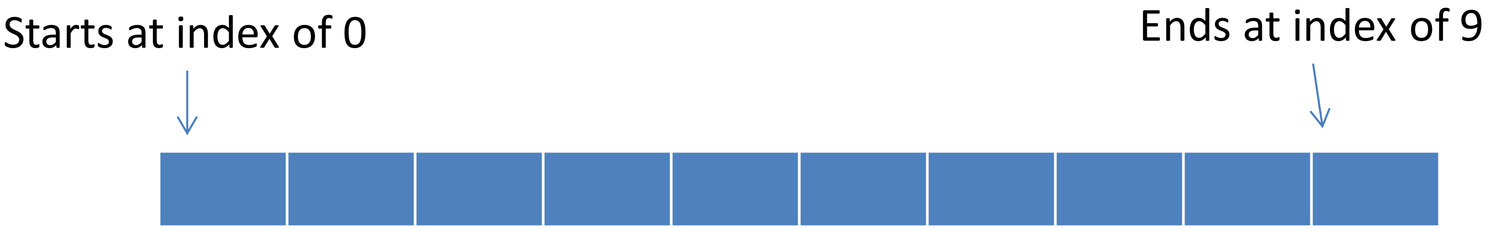
Building Blocks: Lists

- Contiguous set of blocks in memory

```
v = []
for i in range(10):
    v.insert(0, i)
print v
```

Building Blocks: Lists

- 0-based indexing (list of length 10)

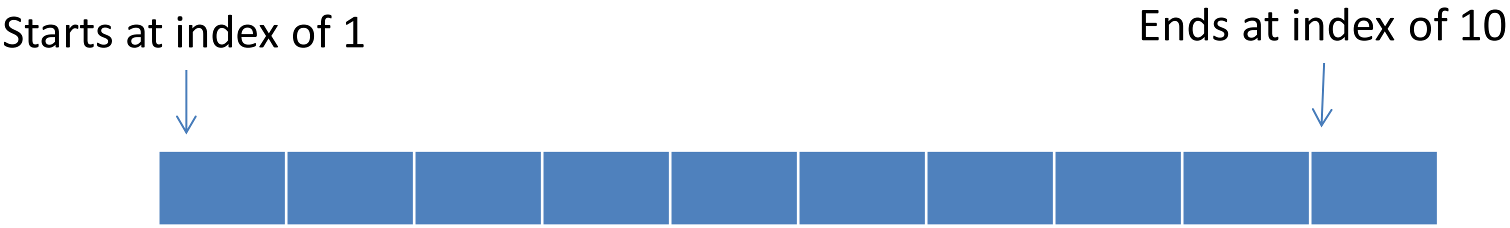


Starts at index of 0

Ends at index of 9

  - Python, C, C++, Java, Javascript

- 1-based indexing (list of length 10)



Starts at index of 1

Ends at index of 10

  - R, MATLAB

Building Blocks: Lists

```
v = [1, 1, 2, 3, 5, 8, 13, 21]
print v.count(1)
print v[0]
print v[-1]
print v[-2]
print v[1:2]
print v[::-1]
print v.index(21)
```

See https://docs.python.org/2/tutorial/datastructures.html for document on lists

Building Blocks: Lists

```
v = [1, 1, 2, 3, 5, 8, 13, 21]
v = v[3:5]
print v
v[0] = 100
print v
v[1] = [1,1,2]
print v
```

Building Blocks: Strings

- Essentially (if not exactly) a list of characters

```
s = "Hello World!"
print s
s = "Hello World!\n"
print s
print s[0:5]
s[0:5] = "HELLO!"
print s
```

Building Blocks: Strings

- Essentially (*if not exactly*) a list of characters

```
s = "Hello World!\n"
s = "HELLO!" + s[5:-1]
print s
```

Building Blocks: Strings

```
s = "Hello World!"
print s.startswith("He")
print s.split()
print s.split("o")
```

Building Blocks: Strings

```
s = "2015"
print s == 2015
print int(s) == 2015
print s == str(2015)
```

Dealing with files and folders

- docs.python.org/2/library/os.path.html
- `import os`
  - (os is a Python module: code that can be imported and used w/ your own programs)
- Get home folder:

```
homeFolder = os.path.expanduser("~")
```

- Get absolute path

```
homeFolder = os.path.abspath(homeFolder)
```

- Append a folder (or file) name to path

```
python2folder = os.path.join(homeFolder, "python_2")
```

Dealing with files and folders

- Make directories if needed:

```
import os, errno
def mkdir_p(path):
    # from http://stackoverflow.com/a/600612
    try:
        os.makedirs(path)
    except OSError as exc: # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else: raise
```

Example: make a folder for today’s lecture

```python
import os, errno

def mkdir_p(path):
    # from http://stackoverflow.com/a/600612
    try:
        os.makedirs(path)
    except OSError as exc: # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else: raise

homeFolder = os.path.abspath(os.path.expanduser("~"))
desktopFolder = os.path.join(homeFolder, "Desktop")
python2folder = os.path.join(desktopFolder, "python_2")
lecture1folder = os.path.join(python2folder, "lecture_1")
print "today's lecture folder location will be:", lecture1folder
mkdir_p(lecture1folder)
```

Downloading a file

```
url = “http://someaddress.com/fileName.txt”

fileName = os.path.basename(url)
fnp = os.path.join(lecture1folder, fileName)
print "going to download", fileName, "from", url

import urllib
urllib.URLopener().retrieve(url, fnp)
```

Reading a large file line-by-line

```
with open(fileNameAndPath) as f:
    for line in f:
        print line
```

Extended Exercise 1

Goal: count how many signal peaks are present in processed ENCODE ChIP-seq data on chromosome 7

url:

bib3.umassmed.edu/~purcarom/Python2/Lecture1/ENCFF002COQ.narrowPeak

File format:

genome.ucsc.edu/FAQ/FAQformat.html#format12

Answer hint: between 2000 and 3000

Extended Exercise 2

Modify code from Extended Exercise 1 to count what percentage of chromosome 7 (assume hg19) is covered by peaks.

Length of chr7 in hg19: 159138663

(Length of HG19 chromosomes in hg19.chrom.sizes)

http://bioinfo.umassmed.edu/bootstrappers/bootstrappers-courses/python2/lecture1/

Building Blocks: Functions

- Functions help divide code up into smaller chunks
  - easier to understand
  - easier to test individual function
  - encourages code reuse
  - can hide details not pertinent to higher-level understanding of code
  - helps introduces structure to code
    - in some ways analogous to English paragraphs
      - i.e. more difficult to read a paper or novel if there were no paragraph breaks or sentences: all the words just run together

Building Blocks: Functions

```
def functionName(parameters):
    code (no more than a “screenful”,
             i.e. approx. <40 lines)
    optionally, return something

def chrNumAsString(num):
    return “chr” + str(num)
```

Download a file if size changed

```
def get_file_if_size_diff(url, d):
    fn = url.split('/')[-1]
    out_fnp = os.path.join(d, fn)
    net_file_size = int(urllib.urlopen(url).info()['Content-Length'])
    if os.path.exists(out_fnp):
        fn_size = os.path.getsize(out_fnp)
        if fn_size == net_file_size:
            print "skipping download of", fn
            return out_fnp
        else:
            print "files sizes differed:"
            print "\t", "on disk:", fn_size
            print "\t", "from net:", net_file_size
    print "retrieving", fn
    urllib.urlretrieve(url, out_fnp)
    return out_fnp
```

Building Blocks: Classes

- In an ideal world, a class contains
  - data (numbers, strings, lists, other classes, etc.)
  - the functions allowed to manipulate the data
- Provides further mechanisms to give code structure
  - Allows data and concepts to be encapsulated (i.e. hidden or only occur in one place)

Building Blocks: Classes

- Classes have constructors or initializers that perform certain operations (like setting up data structures, etc.) when the class is first created
- Python classes have a "`self`" variable that understands how to access data in the class

Building Blocks: Classes

```
class ExampleKlass():
    def __init__(self, strData):
        self.data = strData

    def printData(self):
        print self.data

ek = ExampleKlass(“hi!”)
ek.printData()
```

Classes and Objects

- In object-oriented programming, a “class” is the code that defines
  - the class variables
  - the functions allowed to operate on those class variables
  - No memory taken up
- An “object” is an instance of a class
  - Memory will be allocated for the variables in the object

Class Class Exercise

Make a Paths class that encapsulates the path manipulations we performed earlier

```
homeFolder = os.path.abspath(os.path.expanduser("~"))
desktopFolder = os.path.join(homeFolder, "Desktop")
python2folder = os.path.join(desktopFolder, "python_2")
lecture1folder = os.path.join(python2folder, "lecture_1")
print "today's lecture folder location will be:", lecture1folder
mkdir_p(lecture1folder)
```

Class Class Exercise

```
homeFolder = os.path.abspath(os.path.expanduser("~"))
python2folder = os.path.join(homeFolder, "python_2")
lecture1folder = os.path.join(python2folder, "lecture_1")
print "today's lecture folder location will be:", lecture1folder
mkdir_p(lecture1folder)
```

```
class Paths():
    def __init__(self, lectureNumber):
        self.homeFolder = os.path.abspath(os.path.expanduser("~"))
        self.desktopFolder = os.path.join(self.homeFolder, "Desktop")
        self.python2folder = os.path.join(self.desktopFolder, "python_2")
        lecture = "lecture_" + str(lectureNumber)
        self.lectureFolder = os.path.join(self.python2folder, lecture)
        print "today's lecture folder location will be:", self.lectureFolder
        mkdir_p(self.lectureFolder)

paths = Paths(1)
```

Why use classes?

- What data needed to describe a book?

```
title = ""
author_first_name = ""
author_last_name = ""
year_published = ""
num_pages = ""
```

Why use classes?

- What data needed to describe a book?

```
title = ""
author_1_first_name = ""
author_1_last_name = ""
author_2_first_name = ""
author_2_last_name = ""
year_published = ""
num_pages = ""
```

Why use classes?

- What data needed to describe a book?

```
title = ""
author_1_first_name = ""
author_1_last_name = ""
author_2_first_name = ""
author_2_last_name = "“
...
author_n_first_name = “”
author_n_last_name = “”
year_published = ""
num_pages = "“
front_cover_picture = “”
back_cover_picture = “”
language = “”
edition_number = “”
```

Book class composition

```
class Author:
    def __init__(self):
        self.first_name = ""
        ...

class Edition:
    def __init__(self):
        self.title = ""
        self.authors = [Author(...), ]
        self.isHardcover = True
        ...

class Book:
    def __init__(self):
        self.editions = [Edition(...),]
```

Book class composition

```
books = []
b = Book()
edition1 = Edition(…)
b.editions.append(edition1)
books.append(b)
```

Extended Exercise 3

- Rework your code from the previous Extended Exercises into a ChipseqData class, similar to

```
class ChipseqData:
    def __init__(self, paths, url):
```

- The class initializer should only download the chipseq data if needed
  - What are the pros/cons of downloading the data during class initialization?
- The class should have a function to return the number of peaks found in a given chromosome
- The class should also have a function to compute the percentage of a given chromosome covered by peaks

Homework #1

- Work through these problems from http://rosalind.info/problems/list-view/

- DNA Counting DNA Nucleotides
- RNA Transcribing DNA into RNA
- REVC Complementing a Strand of DNA
- GC Computing GC Content
- HAMM Counting Point Mutations
- SPLC RNA Splicing
- PROT Translating RNA into Protein
- SUBS Finding a Motif in DNA
- PRTM Calculating Protein Mass
- REVP Locating Restriction Sites