Floating Point Arithmetic

- Fields: sign, mantissa, exponent (Lab 7)
  - Issues:
    - Precision and accuracy: $\pi = 3.133333$
    - Rounding and digit cancellation
  - What is $1.00+0.002-0.999$ with a 3-digit mantissa?
    - Audio project (Project 1, Lab 4)
    - Value of $(1-\cos(x))/x^2$ at $x=0$ (sample program)
    - Algorithms for median and variance (Lab 8)
    - Bank rounding, the value of a penny (Lab 7)

Arithmetic Issues

- Round-off error:

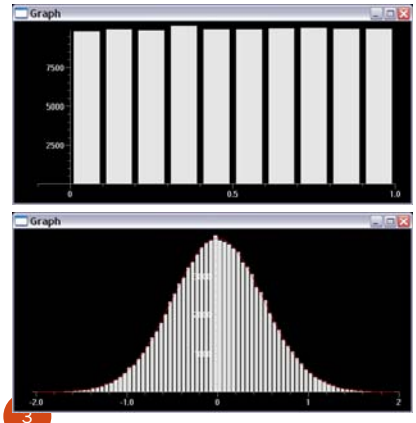
  $0.124 \times 0.351 = 0.043524: \ 0.435 \times 10^{-1}$

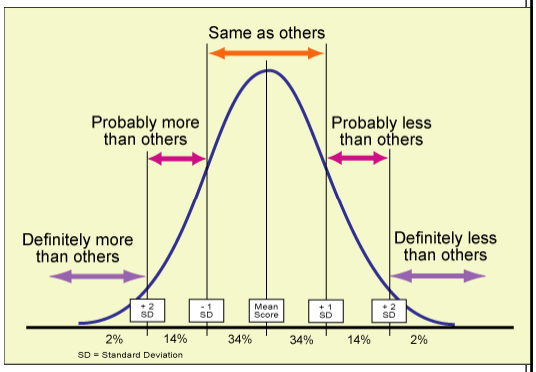
  $\Delta = 0.24 \times 10^{-4}$
- Digit cancellation error:

  $0.127 - 0.124 = 0.003: 0.300 \times 10^{-2}$
- Large summations are problematic, as are iterated computations in general…

Random Numbers

- Uniform and normal distributions

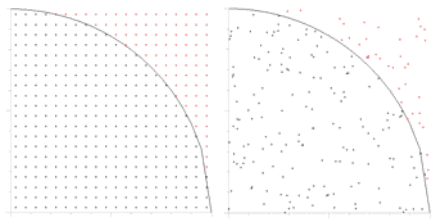


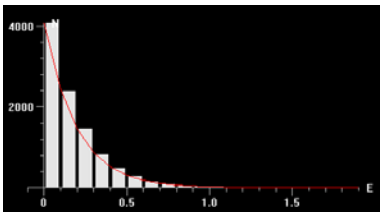


3

Random Numbers

- Pseudo random number generators
  - Quality assessment visually
  - Role of seed value (how about 0?)
  - Don't use $r_{k+1} = (r_k * A + C) \bmod M$ without analysis…
- Monte Carlo and random algorithms
  - p/4 from quarter circle, random vs. regular sampling
  - Demon algorithms





Review Topics

- Binary tree examples
  - Classes
  - Trees

5

Clicker Question (Review)

How do you create an instance of a Python class named "MyClass"?

A. `x = new MyClass()`
B. `x = MyClass()`
C. `MyClass.x(new)`
D. `x.MyClass = new`
E. `x.y`

6

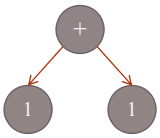
Clicker Question (Review)

How do you call a method named "f" inside object "x"?

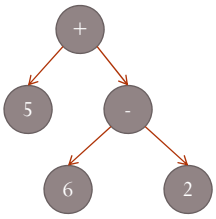
A. `f(x)`
B. `x.f()`
C. `f.x()`
D. `x(f)`
E. `All of the above`

7

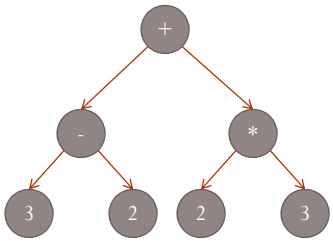
Expressions as (Upside Down) Trees



1 + 1



5 + (6 – 2)



(3 – 2) + (2 * 3)

8

Binary Tree Class (Improved)

```
class BinaryTreeNode:
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

v1 = BinaryTreeNode(1)
v2 = BinaryTreeNode(1)
v = BinaryTreeNode("+", v1, v2)
```

9

Expression Parser

```
def parse(expression):
    stack = []
    for token in expression.split():
        if token.isdigit():
            node = BinaryTreeNode(token)
        else:
            right = stack.pop()
            left = stack.pop()
            node = BinaryTreeNode(token, left, right)
        stack.append(node)
    assert(len(stack) == 1)
    return stack[0]
```

10

Implementation: String method

```
def __str__(self):
    if self.left == None:
        return self.data
    else:
        return "(%s) %s (%s)" %
            (str(self.left), self.data, str(self.right))
```

11

Lab 9 – Version 1

```
from binarytree import *

def postorder(tree, carryover=""):
    if tree.left != None:
        carryover = postorder(tree.left, carryover)
    if tree.right != None:
        carryover = postorder(tree.right, carryover)
    if carryover != "":
        carryover += " " + tree.data
    else:
        carryover = tree.data
    return carryover
```

Parameter "carryover" simplifies blank insertion when building up result string. Other solutions possible.

12

Lab 9 – Version 2

```
from binarytree import *

def postorder(tree):
    if tree.left != None:
        left = postorder(tree.left)
        right = postorder(tree.right)
        return left + " " + right + " " + tree.data
    else:
        return tree.data
```

Simpler, based on assumption that non-leaf nodes have both left and right children.

13

Recursive Function on Binary Tree

- Counts the number of nodes in a tree…

```
def count(node):
      c = 1
      if node.left != None:
          c += count(node.left)
      if node.right != None:
          c += count(node.right)
      return c
```

- How would you convert this function to a method?

14