Numbers, APPROXIMATIONS, and BISECTION

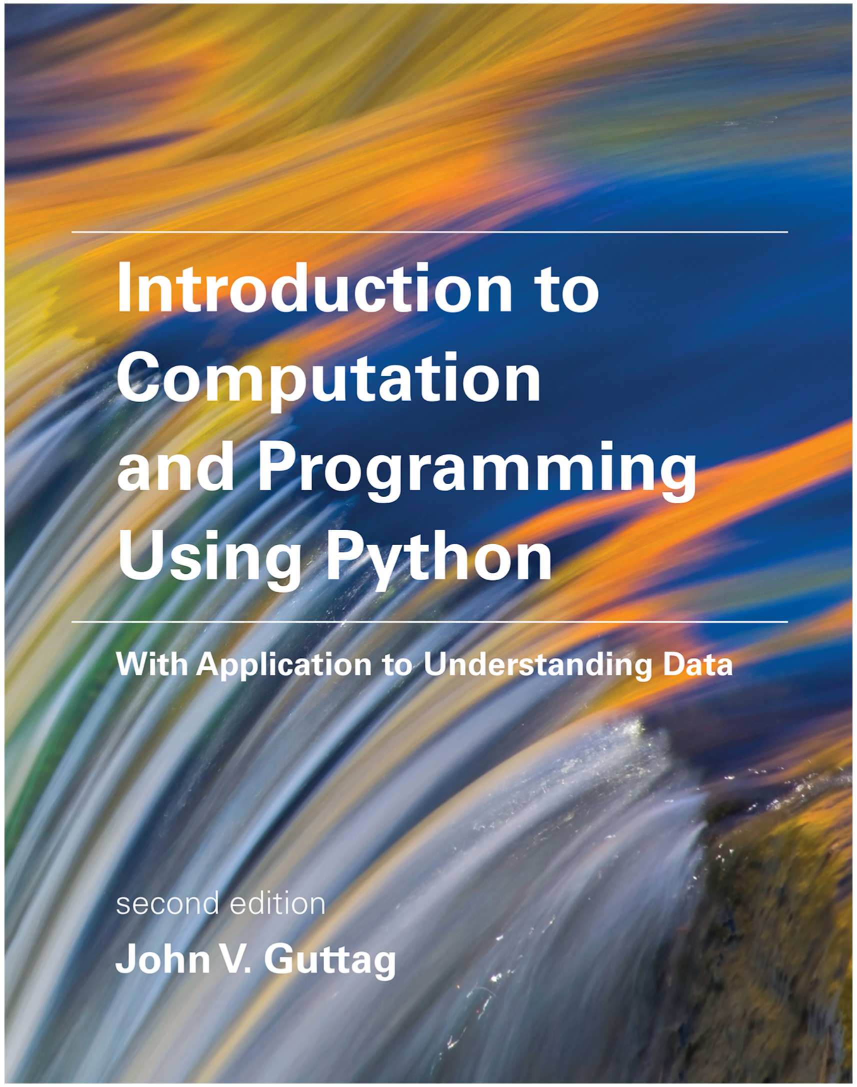
(download slides and .py files from Stellar to follow along)

6.0001 LECTURE 3

John Guttag

Assigned Reading

- Sections 3.3 – 3.5
- Section 4.1


Introduction to Computation and Programming Using Python
With Application to Understanding Data
second edition
John V. Guttag

https://mitpress.mit.edu/sites/default/files/Guttag_errata_revised_083117.pdf

Numbers in Python

- int: integers, like the ones you learned in elementary school
- float: ~~reals, like the ones you learned about in middle school~~

A Closer Look at Floating Point

- Python (and everything else) uses “floating point” to approximate real numbers
- Refers to way they are stored in computer
- Usually doesn’t matter

“Usually doesn’t matter” is another way of saying **?**

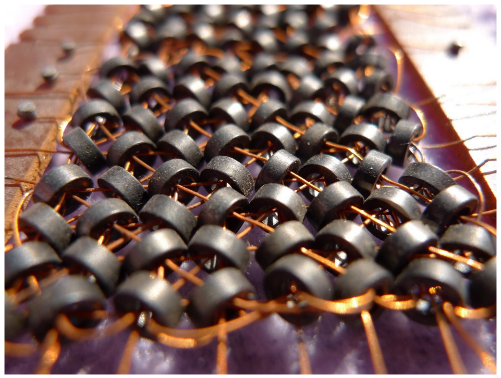
It Matters!

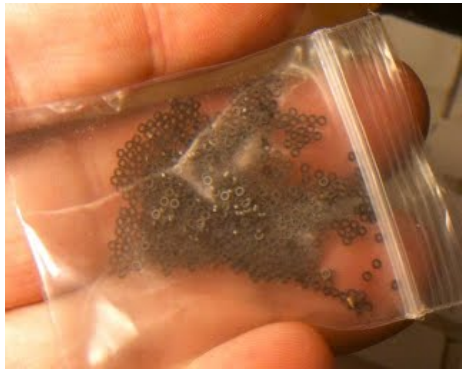
```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)

print(x, '==', 10*0.1)
```

Memory of Modern Computers is Binary

- Representation of floating point numbers is a function of the computer hardware
- In all modern computers, numbers (and everything else) represented as a **sequence of bits** (0 or 1). Think of these as binary numbers (i.e., base 2)
- Easy to implement in hardware—build components that can be in **one** of **two** states

1955 ($1/bit) - 1975 ($0.01/bit)





What would 16GB (what I have on my laptop) have cost in 1955?

Binary Numbers and Floating Point

- Binary number is a sequence of 0’s and 1’s
  - $1001 = 1*2^0 + 0*2^1 + 0*2^2 + 1*2^3 = 9$
- Floating point is a pair of integers
  - Significant digits and exponent
  - $(1, 1) = 1*2^1 = 2.0$
  - $(1, -1) = 1*2^{-1} = 0.5$
- The maximum number of significant digits governs the precision with which numbers can be represented.
  - When exceeded, numbers are rounded

…and i should care, why?

Fractions

- What does the decimal representation 0.125 mean
  - $1*10^{-1} + 2*10^{-2} + 5*10^{-3}$
- Suppose we want to represent it in floating point?
  - $(1, -3) = 1*2^{-3} = 1/8$

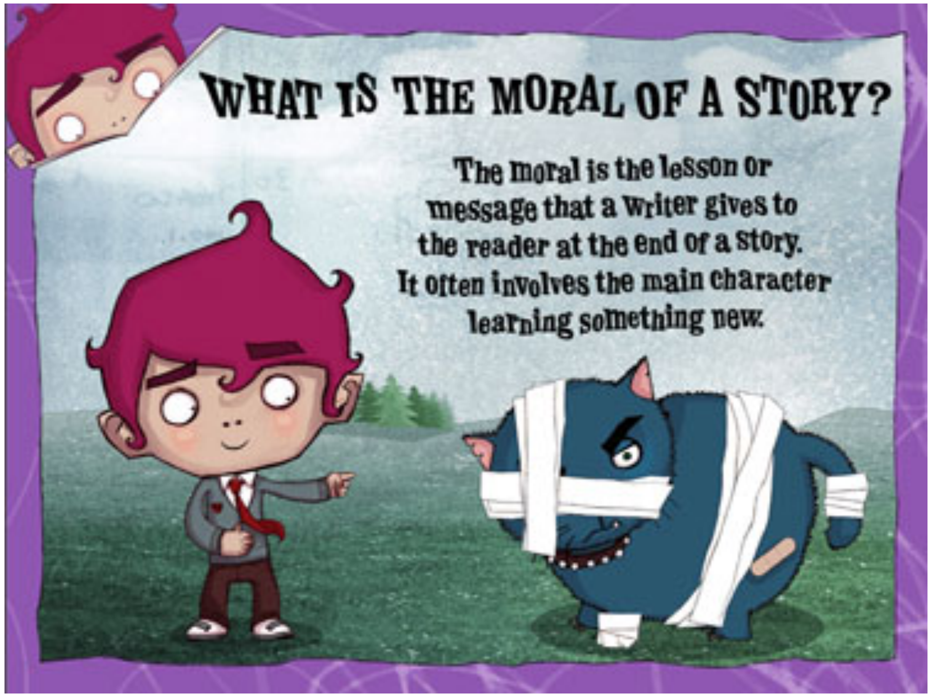
```
x = 0
for i in range(8):
    x += 0.125
print(x == 1)
```

Fractions

- How how about the decimal representation 0.1
  - In base 10: $1 * 10^{-1}$
  - In floating point: (0.0001100110011001100110011…, -101)
- And 10 times that is not 1.0

Any finite number of digits gives us an approximation

The Moral of the Story


WHAT IS THE MORAL OF A STORY?
The moral is the lesson or message that a writer gives to the reader at the end of a story. It often involves the main character learning something new.

```
x = 0
for i in range(10):
    x += 0.1
print(x == 1)
print(x, '==', 10*0.1)
```

Never use == to test floats

Instead test whether they are within epsilon of each other

What gets printed isn't always what is in memory

Finding Roots

- Monday we looked at using exhaustive enumeration/guess and check, to the roots of a perfect square

```
x = int(input("Enter an integer: "))
foundAnswer = False
for guess in range(x + 2):
    if guess**2 == x:
        print("Square root of", x, "is", guess)
        foundAnswer = True
if not foundAnswer:
    print(x, "is not a perfect square")
```

- Suppose we want to find the square root of any positive number
- Question 1: What does it mean to find the square root of x?
  - Find an r such that r*r = x ?

Approximation Algorithms

- Find an answer that is “good enough”
  - E.g., find a r such that r*r is within a given constant of x

- Algorithm
  - Start with guess known to be too small
  - Increment by some small value

- Decreasing increment size → slower program
- Increasing epsilon → less accurate answer

Implementation

```
x = int(input("Enter an integer: "))
epsilon = 0.01
numGuesses = 0
ans = 0.0
increment = 0.0001
while abs(ans**2 - x) >= epsilon:
    ans += increment
    numGuesses += 1
print('numGuesses =', numGuesses)
if abs(ans**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(ans, 'is close to square root of', x)
```

Why do we think that loop will terminate?

Reasoning About Loop Termination

- Decrementing function
  - Function maps variables in program to a number
  - Starts out >= 0
  - Decreased each time loop body is executed
  - Loop is exited when value is <= 0

Is “decreased each time” strong enough to guarantee termination?

Reasoning About Loop Termination

MY CLIENT COULDN'T HAVE KILLED ANYONE WITH THIS ARROW, AND I CAN *PROVE* IT!
I'D LIKE TO EXAMINE YOUR PROOF, ZENO. YOU MAY APPROACH THE BENCH.
—BUT NEVER REACH IT!

Decreased each time in a way that guarantees that it is reaches 0 in a finite number of steps

Approximation Algorithms

```
x = int(input("Enter an integer: "))
epsilon = 0.01
numGuesses = 0
ans = 0.0
increment = 0.0001
while abs(ans**2 - x) >= epsilon:
    ans += increment
    numGuesses += 1
print('numGuesses =', numGuesses)
if abs(ans**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(ans, 'is close to square root of', x)
```

`abs(ans**2 - x) – epsilon`

`Initial value = ?`

`Decremented by ? each iteration`

Approximation Algorithms

```
x = int(input("Enter an integer: "))
epsilon = 0.01
numGuesses = 0
ans = 0.0
increment = 0.0001
while abs(ans**2 - x) >= epsilon:
    ans += increment
    numGuesses += 1
print('numGuesses =', numGuesses)
if abs(ans**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(ans, 'is close to square root of', x)
```

Will test ever return True?

Run it

Some Things to Note

- Try 36
  - Didn't find 6, but that's okay
  - Took about 60,000 guesses
- Try 24, 2, 12345, 54321

Let’s Debug It

```
x = 54321
epsilon = 0.01
numGuesses = 0
ans = 0.0
increment = 0.0001
while abs(ans**2 - x) >= epsilon:
    ans += increment
    numGuesses += 1
    if numGuesses%50000 == 0:
        print('Current guess =', ans)
        print('Current guess**2 - x =',
                ans*ans - x)
print('numGuesses =', numGuesses)
if abs(ans**2 - x) >= epsilon:
    print('Failed on square root of', x)
else:
    print(ans, 'is close to square root of', x)
```

Things to Notice

- Decrementing function incrementing
- We have over-shot the mark
- We didn't account for this possibility when writing the loop
- Let's fix that: while abs(ans**2 – x) >= epsilon and ans**2 <= x:
- Now it stops, but reports failure, because it has over-shot the answer
- Let's look at penultimate guess

```
        print('Previous guess squared was',
              (ans - increment)**2)
```

- Increment too large, skipped over answer
- Let's try resetting increment to 0.00001

How many iterations of loop?

Five Minute Break

Chance to Win a Buck

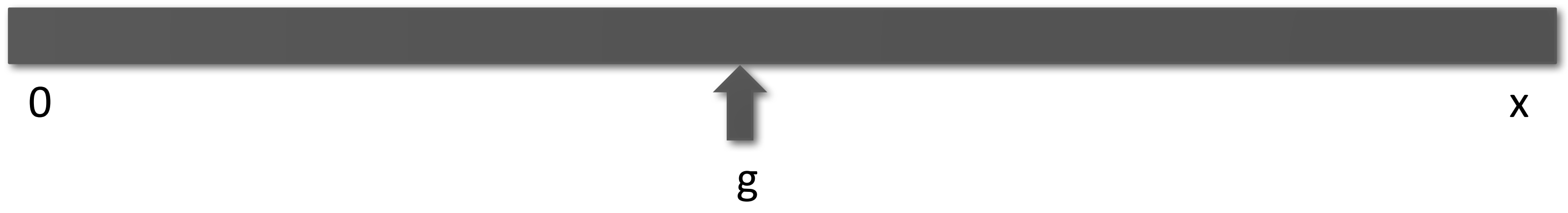
- I have placed a dollar bill after page x in text book
- If you can guess page in 8 or fewer guesses, you get the buck
- If you fail, you lose a late day
- Hint: the book is 447 pages long
- Who wants to play?

Bisection Search

- Suppose we know answer lies within some interval
- Guess midpoint of interval
- If not answer, then check if answer is greater than or less than midpoint
- Change interval
- Repeat
- Process cuts set of things to check in half at each stage

Bisection Search

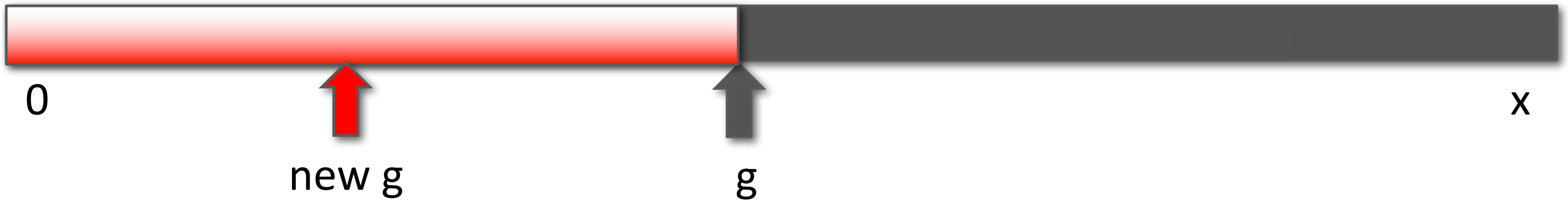
- Suppose we know that the answer lies between 0 and x
- Rather than exhaustively trying things starting at 0, we pick a number in the middle of this range



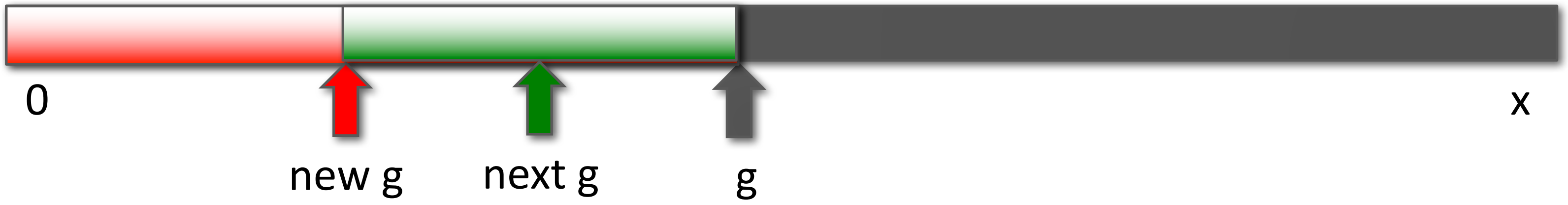
- If we are lucky, this answer is close enough

Bisection Search

- If not close enough, is guess too big or too small?
- If g**2 > x, then know g is too big; but now search



- And if, for example, this new g is such that g**2 < x, then know too small; so now search



- At each stage, reduce range of values to search by half

Replace algorithm that is linear in the number of possible guesses with one is that logarithmic in the number of possible guesses

Fast Square Root

```
x = 54321
epsilon = 0.01
numGuesses = 0
low = 1.0
high = x
ans = (high + low)/2

while abs(ans**2 - x) >= epsilon:
    print('low = ' + str(low) + ' high = ' + str(high)\
          + ' ans = ' + str(ans))
    numGuesses += 1
    if ans**2 < x:
        low = ans
    else:
        high = ans
    ans = (high + low)/2.0
print('numGuesses = ' + str(numGuesses))
print(str(ans) + ' is close to square root of ' + str(x))
```

For what values of x does this work?

x < 1 ?

- If x < 1, could search from 0 to x but square root is greater than x and less than 1
- Modify the code to choose the search space depending on value of x

Some Observations

- Bisection search radically reduces computation time – being smart about generating guesses is important
- Search space gets smaller quickly at the beginning and then more slowly (in absolute terms, but not as a fraction of search space) later
- Works on problems with “ordering” property – value of function being solved varies monotonically with input value
  - Here function is `ans**2`; which grows as `ans` grows
- Not the best way to find roots, Newton-Raphson converges faster
- But bisection search is a general concept that works very well in many contexts