Announcements

- Homework
  - You should be able to do everything on homework 3 after today!
  - All submitted HW2 scores should be in your repository. Look them over before submitting the next assignment!
- CS Tea tomorrow at 11:35 in Ford 202
  - WU Alum David Chidester speaking about Data Engineering
- Polling: `rembold-class.ddns.net`

Review Question

In which of the following situations would using a `for` loop (without a `break` statement) not be recommended?

A) Finding all integers between 1 and 100 which are evenly divisible by 7.

B) Calculating the position of a dropped ball over the first 60 seconds of its flight in 2 second intervals.

C) Finding the fifth prime number greater than 50.

D) Counting the number of capital letters in an arbitrary string.

Pythagorean Integers: For Loop Version

- Let's revisit our problem of finding all positive *integers* a and b such that

$$a^2 + b^2 = 250$$

While Loops

```
a = 0
while 250 - a > 0:
    b = 0
    while 250 - b > 0:
        if a**2 + b**2 == 250:
            print('-----')
            print('A=',a)
            print('B=',b)
        b = b + 1
    a = a + 1
```

For Loops

```
for a in range(250):
    for b in range(250):
        if a**2 + b**2 == 250:
            print('-----')
            print('A=',a)
            print('B=',b)
```

Making the Choice

While Loops:

- Very general and flexible
- Can check and terminate on any condition you can imagine
- You are responsible for initializing and updating needed variables
- Need to be careful of infinite loops
- Can mimic behavior of any for loop

For Loops:

- Only iterate over sequences
- Variable initialization and updating is handled by the sequence
- Impossible to get an infinite loop
- Simpler syntax in general
- Can not mimic every while loop

More Involved Example

Let `s` be a string that contains a sequence of decimal numbers separated by commas. For example, `s = 1.23,4.67,8.37`. Write a program that prints the sum of the numbers in `s`.

When Close Counts

- Computers work with literals and operations, not symbolic mathematics

When Close Counts

- Computers work with literals and operations, not symbolic mathematics
    - Solutions of numeric problem must return actual numbers, either integer or float

When Close Counts

- Computers work with literals and operations, not symbolic mathematics
  - Solutions of numeric problem must return actual numbers, either integer or float
  - Many values can not be represented exactly with a limited number of digits

When Close Counts

- Computers work with literals and operations, not symbolic mathematics
  - Solutions of numeric problem must return actual numbers, either integer or float
  - Many values can not be represented exactly with a limited number of digits
- Need to decide beforehand how exact of a solution to your problem that you need

When Close Counts

- Computers work with literals and operations, not symbolic mathematics
  - Solutions of numeric problem must return actual numbers, either integer or float
  - Many values can not be represented exactly with a limited number of digits
- Need to decide beforehand how exact of a solution to your problem that you need
  - How close is "close enough"?

When Close Counts

- Computers work with literals and operations, not symbolic mathematics
  - Solutions of numeric problem must return actual numbers, either integer or float
  - Many values can not be represented exactly with a limited number of digits
- Need to decide beforehand how exact of a solution to your problem that you need
  - How close is “close enough”?
- Just because approximate, does not mean it is inferior to the “exact” solution!!

Determining “Close Enough”

- Often dependent on the problem

Determining “Close Enough”

- Often dependent on the problem
  - “How old are you?”

Determining “Close Enough”

- Often dependent on the problem
  - “How old are you?”
  - “How long is the blink of an eye?”

Determining “Close Enough”

- Often dependent on the problem
  - “How old are you?”
  - “How long is the blink of an eye?”
- For now, let’s mimic the mathematicians and consider “close enough” to be within some value, $\varepsilon =$ `epsilon`, of the true value

Determining "Close Enough"

- Often dependent on the problem
  - "How old are you?"
  - "How long is the blink of an eye?"
- For now, let's mimic the mathematicians and consider "close enough" to be within some value, $\varepsilon =$ `epsilon`, of the true value
  - "How old are you?"

Determining "Close Enough"

- Often dependent on the problem
  - "How old are you?"
  - "How long is the blink of an eye?"
- For now, let's mimic the mathematicians and consider "close enough" to be within some value, $\varepsilon =$ `epsilon`, of the true value
  - "How old are you?"
    - $\varepsilon \approx 1$ year

Determining "Close Enough"

- Often dependent on the problem
  - "How old are you?"
  - "How long is the blink of an eye?"
- For now, let's mimic the mathematicians and consider "close enough" to be within some value, $\varepsilon =$ `epsilon`, of the true value
  - "How old are you?"
    - $\varepsilon \approx 1$ year
  - Eyeblink

Determining “Close Enough”

- Often dependent on the problem
  - “How old are you?”
  - “How long is the blink of an eye?”
- For now, let’s mimic the mathematicians and consider “close enough” to be within some value, $\varepsilon =$ `epsilon`, of the true value
  - “How old are you?”
    - $\varepsilon \approx 1$ year
  - Eyeblink
    - $\varepsilon \approx 1$ microsecond?

Cubic Example

Example

Suppose we wanted to find the cube root of 27, and didn't know how to invert powers. We require that our answer, when cubed, is within 0.1 of the value 27.

Cubic Example

Example

Suppose we wanted to find the cube root of 27, and didn't know how to invert powers. We require that our answer, when cubed, is within 0.1 of the value 27.

- Might seem like a good `for` loop problem, since we can write down the sequence from 0 to 27 with some step size
  - Careful! The `step` parameter of range must be an integer!
  - Means it will probably be easier to just use a `while` loop

Cubic Example

Example

Suppose we wanted to find the cube root of 27, and didn't know how to invert powers. We require that our answer, when cubed, is within 0.1 of the value 27.

- Might seem like a good `for` loop problem, since we can write down the sequence from 0 to 27 with some step size
  - Careful! The `step` parameter of range must be an integer!
  - Means it will probably be easier to just use a `while` loop
- Introducing some new syntax:
  - `x = x + 1` is the same as `x += 1`
  - `y = y - 2` is the same as `y -= 2`
  - `z = z * n` is the same as `z *= n`
  - And similarly for division, integer division, etc

Example: Ball Drop

Example

The equation that governs the motion of a ball moving purely vertically is:

$$y = y_i + v_0 t - \frac{1}{2} g t^2$$

where

$$\begin{aligned} y_i &= \text{ the initial height} \\ v_0 &= \text{ the initial velocity} \\ g &= \text{ the acceleration due to gravity (9.8 on Earth)} \\ t &= \text{ the elapsed time} \end{aligned}$$

When does a ball which is tossed upwards at $1\,\mathrm{m/s}$ from a height of $2\,\mathrm{m}$ strike the ground $(y = 0)$?

Playing with Approximations

- There are always tradeoffs in choosing `epsilon`
  - The smaller, the more accurate the answer
  - But it will often take *FAR* more time to find that answer
- Usually good for the problem, as it forces you to debate "What is the biggest error I'd find acceptable?"
- But sometimes it is just time to shift to a different, smarter algorithm

Bisection Searches

- Finding a word in a dictionary
  - Open in initial guess
  - Check what letters are you are
  - Make a new guess going in the correct direction
  - Open to the new guess, and repeat

Bisection Searches

- Finding a word in a dictionary
  - Open in initial guess
  - Check what letters are you are
  - Make a new guess going in the correct direction
  - Open to the new guess, and repeat
- Guessing game with higher/lower
  - Guessing a number in the middle
  - Depending on higher or lower, take that new range and guess a number in the middle of that

Bisection Searches

- Finding a word in a dictionary
  - Open in initial guess
  - Check what letters are you are
  - Make a new guess going in the correct direction
  - Open to the new guess, and repeat
- Guessing game with higher/lower
  - Guessing a number in the middle
  - Depending on higher or lower, take that new range and guess a number in the middle of that
- Bisection searches take advantage of the fact that numbers are totally ordered!

Visualizing the Guessing Game

0 100

Visualizing the Guessing Game

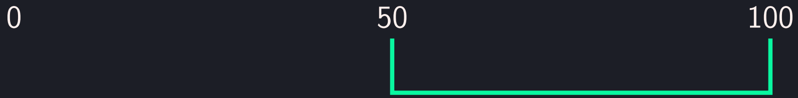
0 50 100

Visualizing the Guessing Game

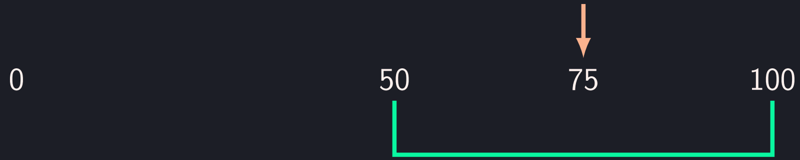
0 50 100

`Your number is too low!`

Visualizing the Guessing Game



Visualizing the Guessing Game

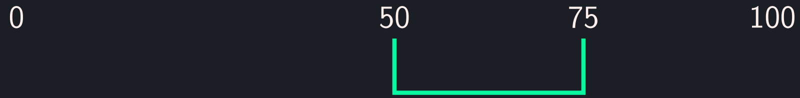


Visualizing the Guessing Game

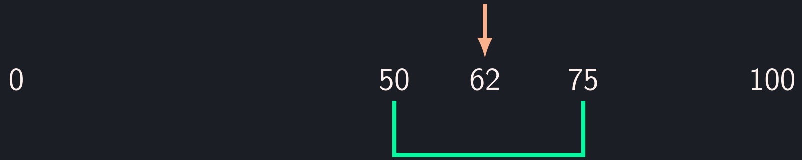
0 50 75 100

`Your number is too high!`

Visualizing the Guessing Game



Visualizing the Guessing Game



Visualizing the Guessing Game

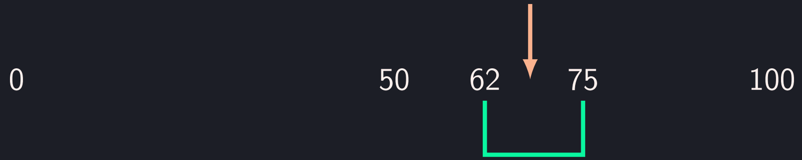
0 50 62 75 100

`Your number is too low!`

Visualizing the Guessing Game

0 50 62 75 100

Visualizing the Guessing Game



Bisection Requirements

- You need to have known edge points to begin!
  - What is initially the low end of your range?
  - What is initially the high end of your range?
- You must update your low and high ends according to how your guess compares to the actual value
  - If guess is too low, then that guess becomes the new low end of the range
  - If guess is too high, then guess becomes the new high end of the range
- Take your new guess to be the average of the low and high points
  - Gets you a point exactly in the middle
  - Lets you best leverage the information provided by the number ordering

Bisection Examples

Example

Let's implement a bisection search in our program to find the cube root of 27. How does the number of iterations required to find an answer compare?

Bisection Examples

Example

Let's implement a bisection search in our program to find the cube root of 27. How does the number of iterations required to find an answer compare?

Example

Let's write a program to try to guess an unknown number between 1 and 100. Essentially, let's write a program to play the game you wrote on HW2.