there is a new drink called the Sandy, it is a watered down Manhattan.
too soon?

Like · Comment · 11 minutes ago near

8 people like this.

Write a comment...

HURRICANE SANDY @AHurrica.2h RETWEET IF U WANT ME 2 DESTROY YO SCHOOL. I GOT YOU.

HURRICANE SANDY @AHurrica.1h .@MITTROMNEY YOU IS A BASIC BITCH.

HURRICANE SANDY @AHurrica.1h 50 SHADES OF ME F**KING UP THE EAST COAST.

HURRICANE SANDY @AHurrica.31m I WENT TO HIGHSCHOOL WIT IRENE. SHE CAN'T EVEN TWERK. SHE AIN'T BOUT DAT HURRICANE LYFE.

HURRICANE SANDY @AHurrica.20m JUS BLEW DA ROOF OFF A OLIVE GARDEN FREE BREADSTICKS FOR EVERYONE
New Schedule

Work to do.

Time to do.

What we do!
Lab 6:  
avg=95/100

1. Handling floats, weird output for negative/float numbers
   Please give a value of n:-2
   The sum of the first -2 natural numbers is: 0
   The sum of the cubes of the first -2 natural numbers is: 0

3. Show off what works: ints, floats, negative, formulas, fractions, different length lists
Lab 7:
Avg = 94/100

1. Testing conditions
   – 30, 9
   – 30, 7
   – 29, 9
   – 30, 8
   – 25, 6
   – 24, 7
Lab 7:
Avg = 94/100

2. 4 combinations
   - 2001: N
   - 1900: N
   - 2004: Y
   - 2000: Y
Lab 7: 
Avg = 94/100 

3. Conjecture 
   – Shouldn’t give answer for 100 or odd numbers. 
   – Don’t print out same numbers twice. 
   – Say something if input was invalid. 
   – Can use list of primes, not calc 
   – Testing Conditions 
     • 0 
     • 9 
     • 10 
     • 98 
     • 100
Homework Questions?

I think you've confused me with someone who builds a dam.
Sentinel Loops

# average4.py
# A program to average a set of numbers
# Illustrates sentinel loop using empty string as sentinel

def main():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")
    print("\nThe average of the numbers is", sum / count)
def main():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split(",",):
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print("\nThe average of the numbers is", sum / count)
Computing with Booleans

• *if* and *while* both use Boolean expressions

• Boolean expressions evaluate to *True* or *False*

• So far, used Boolean expressions to compare two values, e.g.,
  
  (while x >= 0)
Boolean Operators

• Sometimes simple expressions not expressive enough
• Suppose you need to determine whether two points are in the same position – their $x$ coordinates are equal and their $y$ coordinates are equal
Boolean Operators

```python
if p1.getX() == p2.getX():
    if p1.getY() == p2.getY():
        # points are the same
    else:
        # points are different
else:
    # points are different
```

• Clearly, this is awkward
• Boolean operators: `and`, `or`, `and not`
Boolean Operators

• The Boolean operators `and` and `or` are used to combine two Boolean expressions and produce a Boolean result.

• `<expr> and <expr>`

• `<expr> or <expr>`
Boolean Operators

• The and of two expressions is true exactly when both of the expressions are true
• We can represent this in a truth table

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$P$</td>
<td>$Q$</td>
<td>$P$ and $Q$</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Boolean Expressions

• In the truth table, $P$ and $Q$ represent smaller Boolean expressions.
• Since each expression has two possible values, there are four possible combinations of values.
• The last column gives the value of $P$ and $Q$. 
Boolean Expressions

- The $\texttt{or}$ of two expressions is true when either expression is true

<table>
<thead>
<tr>
<th>$P$</th>
<th>$Q$</th>
<th>$P \texttt{or} Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Boolean Expressions

• The only time or is false is when both expressions are false.

• Also, note that or is true when both expressions are true. This isn’t how we normally use “or” in language.
Boolean Operators

- The `not` operator computes the opposite of a Boolean expression.
- `not` is a *unary* operator, meaning it operates on a single expression.

<table>
<thead>
<tr>
<th>$P$</th>
<th>not $P$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Boolean Operators

• Can put these operators together to make complex Boolean expressions
• Interpretation of the expressions relies on the precedence rules for the operators
Boolean Operators

• Consider
  \[ a \text{ or not } b \text{ and } c \]
• How should this be evaluated?
• The order of precedence, from high to low, is
  \[ \text{not and or} \]
• This statement is equivalent to
  \[ (a \text{ or } ((\text{not } b) \text{ and } c)) \]

• Most people (including Matt) don’t memorize the
  the Boolean precedence rules... use parentheses to
  prevent confusion
Boolean Operators

• To test for the co-location of two points, we could use an `and`

```python
if p1.getX() == p2.getX() and p2.getY() == p1.getY():
    # points are the same
else:
    # points are different
```

• Entire condition will be true only when both of the simpler conditions are true
Clicking in a Box

• Suppose you wanted to tell if a user clicked inside the box shown.

• Complete the following code:

```python
p = win.getMouse()
if
    print("Clicked")
else:
    print("You missed")
```
Clicking in a Box

- Well, that’s peachy, but kind of specific
- Write a general method that will tell you if some box has been clicked
- Assume you know $x_{\text{min}}$, $x_{\text{max}}$, $y_{\text{min}}$, $y_{\text{max}}$ of any box you’d want to test
Boolean Operators

• Racquetball simulation: *The game is over as soon as either player has scored 15 points.*

• How can you represent that in a Boolean expression?

   \[
   \text{scoreA} == 15 \text{ or scoreB} == 15
   \]

• When either of the conditions becomes true, the entire expression is true. If neither is true, expression is false
Boolean Operators

• *We want to construct a loop that continues as long as the game is not over*

• You can do this by taking the negation of the game-over condition as your loop condition!

```python
while not(scoreA == 15 or scoreB == 15):
    #continue playing
```
Boolean Operators

- Sometimes, use a shutout rule: *if one player has scored 7 points and the other person hasn’t scored yet, the game is over*

```python
while not(scoreA == 15 or scoreB == 15 or (scoreA == 7 and scoreB == 0) or (scoreB == 7 and scoreA == 0)):
    # continue playing
```
Boolean Operators

• What about volleyball? To win, a team needs to win by at least two points
• A team wins at 15 points
• If the score is 15 – 14, play continues, just as it does for 21 – 20
  • Write down the expression to determine when play should stop
Boolean Operators

- What about volleyball? To win, a team needs to win by at least two points.
- A team wins at 15 points.
- If the score is 15 – 14, play continues, just as it does for 21 – 20.

\[(a \geq 15 \text{ and } a - b \geq 2) \text{ or } (b \geq 15 \text{ and } b - a \geq 2)\]

\[(a \geq 15 \text{ or } b \geq 15) \text{ and } \text{abs}(a - b) \geq 2\]
Boolean Algebra

• The ability to formulate, manipulate, and reason with Boolean expressions is an important skill.

• Boolean expressions obey certain algebraic laws called *Boolean logic* or *Boolean algebra*.
Boolean Algebra

<table>
<thead>
<tr>
<th>Algebra</th>
<th>Boolean algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \times 0 = 0$</td>
<td>$a$ and false == false</td>
</tr>
<tr>
<td>$a \times 1 = a$</td>
<td>$a$ and true == $a$</td>
</tr>
<tr>
<td>$a + 0 = a$</td>
<td>$a$ or false == $a$</td>
</tr>
</tbody>
</table>

- and has properties similar to multiplication
- or has properties similar to addition
- 0 and 1 correspond to false and true, respectively
Boolean Algebra

• Anything or ed with true is true:
  \[ a \text{ or true} \Rightarrow true \]

• Both and and or distribute:
  \[ a \text{ or (b and c)} \Rightarrow (a \text{ or b}) \text{ and (a or c)} \]
  \[ a \text{ and (b or c)} \Rightarrow (a \text{ and b}) \text{ or (a and c)} \]
Boolean Algebra

• Double negatives cancel out:
  \[ \text{not}(\text{not } a) == a \]

• De Morgan’s laws:
  \[ \text{not}(a \text{ or } b) == (\text{not } a) \text{ and } (\text{not } b) \]
  \[ \text{not}(a \text{ and } b) == (\text{not } a) \text{ or } (\text{not } b) \]

Augustus De Morgan
Born: 1806
Died: 1871
Causing pain for students: Today
Boolean Algebra

• We can simplify our Boolean expressions

while not(scoreA == 15 or scoreB == 15):
    #continue playing

• “While it is not the case that player A has 15 or player B has 15, continue playing”
Boolean Algebra

• “While it is not the case that player A has 15 or player B has 15, continue playing”

• Applying DeMorgan’s law:
  while (not scoreA == 15) and (not scoreB == 15):
   # continue playing

  while scoreA != 15 and scoreB != 15
   # continue playing

• “While player A has not reached 15 and player B has not reached 15, continue playing”
Boolean Algebra

- Organizing expressions can make things easier to understand and debug

```python
while not(scoreA == 15 or scoreB == 15):
    # continue playing

while (not scoreA == 15) and (not scoreB == 15):
    # continue playing

while scoreA != 15 and scoreB != 15
    # continue playing
```
Other Common Structures

• The *if* and *while* can be used to express every conceivable algorithm
• For certain problems, an alternative structure can be convenient
Post-Test Loop

• Write a program to get a nonnegative number from user
• If user types an incorrect input, the program asks for another value
• Process continues until a valid value has been entered
• input validation
Post-Test Loop

repeat
  get a number from the user
until number is >= 0