Improving the efficiency of primality testing
Quit the loop when compositeness is detected

- As soon as we discover that $n$ is composite, we are done and we should quit the loop and produce output.

- While this does not improve the *worst case efficiency* of the program, it does improve the typical case.

- We’ll see two ways of doing this.
The break statement

- The **break** statements forces the program to exit out of the smallest enclosing **while**-loop (or **for**-loop).

### Example:

```python
n = 10
while n < 20:
    if n % 7 == 0:
        break
    n = n + 1
print n
```
import math
number = int(raw_input("Enter a positive integer: "))

factor = 2
isPrime = True
factorBound = math.sqrt(n)
while(factor <= factorBound):
    if(number % factor == 0):
        isPrime = False
        break
    factor = factor + 1

if(isPrime):
    print number, "is prime"
else:
    print number, "is composite"
A simple way to time Python programs

- The `time` module contains a bunch of functions that help us time code fragments.

- Calling `time.time()` returns the time elapsed (usually in seconds) since some epoch (maybe Jan 1\textsuperscript{st}, 1970).

- Call `time.time()` twice, once before and once after the code fragment and take the difference.
Another approach

- We want to stay in the loop while

\[ \text{n} \leq \text{factorBound} \text{ (there are more factors to consider) AND isPrime} = \text{True} \text{ (we have not yet found a factor)} \]

- We can express this using the boolean operator `and` in Python.
import math
number = int(raw_input("Enter a positive integer: "))

factor = 2
isPrime = True
factorBound = math.sqrt(n)
while(factor <= factorBound) and (isPrime):
    if(number % factor == 0):
        isPrime = False
    factor = factor + 1

if(isPrime):
    print number, "is prime"
else:
    print number, "is composite"
Python boolean operators

• **and**, **or**, and **not** are the three Python boolean operators

• A and B is true only when both A and B are true.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A and B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Examples: play with these

- $(x \leq 10)$ and $(x > 4)$
- $(x < 4)$ and $(x > 10)$
- $(x < 10)$ and True
- $(x \geq 0)$ and False
The or operator

A or B is true when A is true or B is true (or both).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A or B</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Examples: play with these

- \((x \leq 10) \text{ or } (x > 4)\)
- \((x < 4) \text{ or } (x > 10)\)
- \((x < 10) \text{ or } \text{True}\)
- \((x \geq 0) \text{ or } \text{False}\)
The \texttt{not} operator

<table>
<thead>
<tr>
<th>( A )</th>
<th>( \texttt{not} \ A )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
</tr>
</tbody>
</table>

**Examples:**

- \( \texttt{not} (x < 10) \)
- \( \texttt{not} (x == 10) \)
- \( \texttt{not} (x >= -10) \)