One More Version of the Primality Testing Program
Some programming “purists” think that the use of the `break` statement is bad programming practice.

Comment from an online discussion on programming:

*Generally, breaking out of loops is considered bad form because it tends to obfuscate your code. It's harder to follow the "flow" of a program with `continue/break` thrown in everywhere. It's especially worse if you use it in nested loops, etc.*

I don’t think using the break statement is bad programming practice, but yes it needs to be used with caution.
An alternative to using break

- We want to stay in the loop while
  
  \[ n \leq \text{factorUpperBound} \]
  (there are more factors to consider)
  
  \textbf{and}
  
  \[ \text{isPrime} == \text{True} \]
  (we have not yet found a factor)

- We can express this using the boolean operator and in Python.
import math

n = int(input("Please type a positive integer, greater than 1: "))

factor = 2 # initial value of possible factor
isPrime = True # variable to remember if n is a prime or not
factorUpperBound = math.sqrt(n) # the largest possible factor we need to test is sqrt(n)

# loop to generate and test all possible factors
while (factor <= factorUpperBound) and (isPrime):
    # test if n is evenly divisible by factor
    if (n % factor == 0):
        isPrime = False
    factor = factor + 1

# Output
if isPrime:
    print n, " is a prime."
else:
    print n, " is a 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

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

- A or B is True when A is True or B is True or both.
- In other words, A or B is False only when both A and B are False.

<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 not operator

- This is a *unary* operator, i.e., it operates on only one operand.

<table>
<thead>
<tr>
<th>A</th>
<th>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:**
  - not (x < 10)
  - not (x == 10)
  - not (x>=-10)
The importance of primality testing

- From time to time you may hear in the news about the new largest prime
- Large primes are the basis of modern day cryptography.
- Cryptography is the mathematical and computational study of how to encode a message so that only the intended receiver can understand the message.
- Without cryptography online business (think Amazon, eBay, etc.) would not be possible.
Final remarks on primality testing

- In the *worst case*, the while-loop in the programs makes $\sqrt{n}$ iterations.

- For an input with, say 100 digits, what might the running time be?

- $n = 10^{100}$. Therefore $\sqrt{n} = 10^{50}$. Even if each iteration of the while-loop took a nanosecond ($10^{-9}$ seconds), the program would take $3.17 \times 10^{33}$ years!
The time module contains functions that allow us to determine (within the program), how much time different blocks of code take.

```python
import time
...
start = time.time()
...
# code you want timed
...
end = time.time()
elapsedTime = end - start
```

Try this out to determine how much difference (if any) our improvement to the primality testing program makes.
So how are numbers with 300 digits tested?

- Based on facts in *number theory* (an area of mathematics), several fast primality-testing algorithms have been developed.

- **Examples:**

  *Miller-Rabin test:*

  This is a *randomized* algorithm – a step in the algorithm performed by rolling dice.

  The algorithm is not always correct! A composite number may be classified a prime, with small and tune-able error probability.