Improving our First Program

WEDNESDAY, JAN 26TH
How do **while** statements affect program flow?

Flow
- Line 1, bool expr, Line 2, Line 3, bool expr, Line 2, Line 3, ...
- bool expr, Line 4

Diagram:
- Line 1
- while boolean expression:
  - Line 2
  - Line 3
  - Line 4
- Is boolean expression true?
  - yes
  - Line 2
  - Line 3
  - Line 4
- no
Line 1
while boolean expression:
    Line 2
    Line 3
    Line 4

- Lines 2 and 3 form the *body* of the while loop
- Python uses indentation to identify the lines following the while statement that constitute the body of the while loop.
Boolean expressions

- Python has a type called `bool`.

- The constants in this type are `True` and `False`.  
  (Not `true` and `false`!)

- The comparison operators:
  
  `<    >    <=    >=    !=    ==`

  can be used to construct *boolean expressions*, i.e., expressions that evaluate to *True* or *False*. 
Boolean expressions: examples

- Suppose $x$ has the value 10

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &lt; 10$</td>
<td>False</td>
</tr>
<tr>
<td>$x \neq 100$</td>
<td>True</td>
</tr>
<tr>
<td>$x \leq 10$</td>
<td>True</td>
</tr>
<tr>
<td>$x &gt; -10$</td>
<td>True</td>
</tr>
<tr>
<td>$x \geq 11$</td>
<td>False</td>
</tr>
</tbody>
</table>
A silly while loop example

```python
n = int(raw_input("Enter a positive integer:"))
while n != 0:
    n = n - 2
```

- What happens when input is 8?
- What happens when the input is 9?

The biggest danger with while loops is that they may run forever.
Improving the output

- How can we put together the bits we generate, in the correct order, to construct the binary equivalent?

- **String concatenation!**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>“0” + “1001”</td>
<td>“01001”</td>
</tr>
<tr>
<td>“1” + “1001”</td>
<td>“11001”</td>
</tr>
</tbody>
</table>
Algorithmic idea

- After $i$ iterations of the while loop we have generated the right most $i$ bits of our answer.

- Call this the \textit{length}-$i$ suffix.

- We want to maintain a string:
## Example

- Input is 39.

<table>
<thead>
<tr>
<th>Output</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“”</td>
</tr>
<tr>
<td>1</td>
<td>“1”</td>
</tr>
<tr>
<td>1</td>
<td>“11”</td>
</tr>
<tr>
<td>0</td>
<td>“111”</td>
</tr>
<tr>
<td>0</td>
<td>“0111”</td>
</tr>
<tr>
<td>0</td>
<td>“00111”</td>
</tr>
<tr>
<td>1</td>
<td>“100111”</td>
</tr>
</tbody>
</table>
n = int(raw_input("Enter a positive integer:"))
suffix = ""
while n > 0:
    suffix = str(n % 2) + binary
    n = n/2
print suffix
Making the program more robust

- What if the user types in a negative integer or 0? Or a real number? Or some non-numeric string, (e.g., “hello”)?

- We will only discuss the negative integer or 0 situation now.

- Later when we discuss exceptions and how to handle them, we’ll return to this program.