Our First Programming Problem

JAN 25TH 2013
Problem: Converting decimal numbers to binary

- Given a non-negative integer, convert it into its binary equivalent.

**Example:**
- **Input:** 123 **Output:** 1111011
- **Input:** 1363 **Output:** 10101010011
- **Input:** 12 **Output:** 1100
Plan of Action

1. Understand the problem. What does “binary equivalent” mean?

2. Design an *algorithm* for the problem. How would we solve the problem with a pencil and paper?

3. Write down *pseudocode* for the algorithm.

4. Translate the pseudocode into *Python code*.

5. Think about correctness and test.

6. Think about efficiency. Is the algorithm too slow?
This example will illustrate...

- Constants
- Variables
- Operators
- Data types
- Expressions
- Function calls
- Input statements
- Output statements
- Control flow statements
Consider the decimal number 8,374.

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

Therefore, the “value” of this number is

\[ 8 \times 1000 + 3 \times 100 + 7 \times 10 + 4 \times 1 \]
Similarly, consider the binary number 10110110.

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

Place values: 128 64 32 16 8 4 2 1

Just like the place values for decimal numbers are powers of 10, the place values for binary numbers are powers of 2.

Therefore, the “value” of this number is

\[ 128 + 32 + 16 + 4 + 2 = 182 \]
<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
</tbody>
</table>
Two observations based on this table

**Observation 1:**
If $n$ is even, then its binary equivalent ends with a 0; otherwise, if $n$ is odd, its binary equivalent ends with 1.

(Can you prove this?)
Two observations based on the table

Observation 2:
Suppose that the binary equivalent of \( n \) is 
\[ b_k \ldots b_2 b_1 b_0. \]
If \( n \) is even, then the binary equivalent of \( n/2 \) is 
\[ b_k \ldots b_2 b_1 \]
and if \( n \) is odd, then the binary equivalent of \( (n-1)/2 \) is 
\[ b_k \ldots b_2 b_1. \]

(Can you prove this?)
This suggests an algorithm

1. Check if the given number n is odd or even.

2. If n is even, we know that its binary equivalent ends with 0. Furthermore, to get the rest of n’s binary equivalent, we need to “process” n/2.

3. If n is odd, we know that the binary equivalent ends with 1. Furthermore, to get the rest of n’s binary equivalent, we need to “process” (n-1)/2.
What is an algorithm?

- An algorithm is a step-by-step procedure to complete a task.

- **Examples of algorithms:**
  - A recipe for baking muffins,
  - The output produced by Google maps when you ask for directions from Iowa City to Santa Fe,
  - The procedure for computing the binary equivalent of a decimal integer described in the previous slide.

- The oldest example of a computational algorithm: the 2300-year old *Euclid’s algorithm* for computing the greatest common divisor.

- Your digital life depends on algorithms: web search algorithms, cryptography algorithms, data compression algorithms, etc.
Let the given input be \( n = 203 \).

1. \( n = 203 \) is odd. So rightmost bit is 1.  
   To get the rest of the answer we should “process” \( (n-1)/2 = 101 \).
2. \( n = 101 \) is odd. So the rightmost bit is 1.  
   To get the rest of the answer we should “process” \( (n-1)/2 = 50 \).
3. \( n = 50 \) is even. So the rightmost bit is 0.  
   To get the rest of the answer we should “process” \( n/2 = 25 \).
4. \( n = 25 \) is odd. So the rightmost bit is 1.  
   To get the rest of the answer we should “process” \( (n-1)/2 = 12 \).
5. \( n = 12 \) is even. So the rightmost bit is 0.  
   To get the rest of the answer we should “process” \( n/2 = 6 \).
6. \( n = 6 \) is even. So the rightmost bit is 0.  
   To get the rest of the answer we should “process” \( n/2 = 3 \).
7. \( n = 3 \) is odd. So the rightmost bit is 1.  
   To get the rest of the answer we should “process” \( (n-1)/2 = 1 \).
8. \( n = 1 \) is odd. So the rightmost bit is 1.  
   To get the rest of the answer we should “process” \( (n-1)/2 = 0 \).

So the output (right to left) is 1 1 0 1 0 0 1 1.
Pseudocode

1. Read the number $n$ given as input.
2. If $n$ is even, output 0. Replace $n$ by $n/2$.
3. If $n$ is odd, output 1. Replace $n$ by $(n-1)/2$.
4. If $n$ is 0, stop. Otherwise go to Line 2.

Note that this algorithm produces the binary equivalent of $n$ in “right to left order.”
What is pseudocode?

- Pseudocode is a “language” used to describe algorithms.
- It is not as precise as actual programming language code.
- But it is precise enough that we can reason about correctness and efficiency of the algorithm.
n = int(raw_input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2