Understanding our first program

JAN 27TH 2014
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
n = int(raw_input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2
n = int(raw_input("Type a nonnegative integer:"))

**Assignment statement**

- `=` is the **assignment operator**
- n is a **variable**
- The stuff on the right hand side is an **expression** that gets **evaluated** and its value gets **assigned** to the variable n
Examples of assignment statements

- \( n = 9 \)
- \( n = n/2 \)
  (Assignment operator is not algebraic equality)
- \( n = n + 1 \)
  (A commonly used assignment statement for incrementing the variable \( n \).)
- \( m = n \% 2 \)
  (\( m \) gets assigned 1 if \( n \) is odd; otherwise \( m \) gets assigned 0.)
- \( m = n/5 \)
  (Try out this assignment with \( n \) set to 11 and then with \( n \) set to 11.0).
The `raw_input` function

`raw_input(prompt)`

- This function is a built-in Python function and is always available.

- The `prompt` is written to output (screen) and then the function reads a line from input (keyboard) and returns what it reads.

- `prompt` is an argument to the function `raw_input`.
Functions in Python

- When you are first taught (mathematical) functions in school, you are told to view them as *input-output machines*.
- This is a useful view for functions in Python also.
- The programmer *calls* a function with appropriate inputs, called *arguments* and the function does something (we may not know what) and produces an output.
- In Python, functions can be *built-in* (e.g., `raw_input()`) or *user defined*. 
Try this code snippet. What happens?

```
x = raw_input("Enter a number:")
x = x + 1
```

What the user types is read in as a string, the expression on the right hand side evaluates to a string and \( x \) gets assigned a string.
Every object (e.g., constants, variables) in Python has a *type*

An object’s type determines what operations can be performed on that object.

Use the Python built-in function `type` to determine an object’s data type.
Data types in Python

- Examples:

  Constant
  “Enter a number”
  1034
  55.0

- Python has many built-in data types. For now we will work with four types:

<table>
<thead>
<tr>
<th>type</th>
<th>Python’s type name</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>int</td>
</tr>
<tr>
<td>string</td>
<td>str</td>
</tr>
<tr>
<td>floating point</td>
<td>float</td>
</tr>
<tr>
<td>boolean</td>
<td>bool</td>
</tr>
</tbody>
</table>
The type of a variable is the type of what it was most recently assigned.

**Example:**

```python
x = 15
print(type(x))  # int
x = x*1.0
print(type(x))  # float
```

This ability of the same variable to have different types within a program is called *dynamic typing*.
The meaning of operators (e.g., +, /) depends on the data types they are operating on.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/2</td>
<td>4</td>
<td>int</td>
</tr>
<tr>
<td>9.0/2</td>
<td>4.5</td>
<td>float</td>
</tr>
<tr>
<td>9/2.0</td>
<td>4.5</td>
<td>float</td>
</tr>
<tr>
<td>5 + 1</td>
<td>6</td>
<td>int</td>
</tr>
<tr>
<td>5 + 1.0</td>
<td>6.0</td>
<td>float</td>
</tr>
<tr>
<td>&quot;hello,&quot;+&quot; friend&quot;</td>
<td>&quot;hello, friend&quot;</td>
<td>string</td>
</tr>
</tbody>
</table>
Conversions between data types

- Python provides built-in functions for converting between data types.

**Examples:**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>int(&quot;320&quot;)</td>
<td>320</td>
</tr>
<tr>
<td>float(&quot;320&quot;)</td>
<td>320.0</td>
</tr>
<tr>
<td>str(134)</td>
<td>&quot;134&quot;</td>
</tr>
</tbody>
</table>
n = int(raw_input("Enter a positive integer:"))

1. raw_input prints the prompt, reads a line of the user’s input, and returns what is read as a string.

2. This string gets converted to an integer by the function int.

3. This integer gets assigned to the variable n.
<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 + (12/2.0)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>“12” + “0”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int(“200“)/10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(float(12)/5) + 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>str(25/4) + “00“</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9876 % 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>str(9876 % 100)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(12/5.0) + (12/5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You’ll get more practice in the discussion section and in Practice Problem Set 2 and in Homework 1.
On while-loops

- While-loops affect the flow of the program, i.e., the order in which program statements are executed.

- For the above example the flow of the program is:

  Line 1, bool expr (True), Line 2, Line 3, bool expr (True), Line 2, Line 3, bool expr (False), Line 4
Body of while loop

- 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>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x &lt; 10$</td>
<td>False</td>
<td>bool</td>
</tr>
<tr>
<td>$x != 100$</td>
<td>True</td>
<td>bool</td>
</tr>
<tr>
<td>$x &lt;= 10$</td>
<td>True</td>
<td>bool</td>
</tr>
<tr>
<td>$x &gt; -10$</td>
<td>True</td>
<td>bool</td>
</tr>
<tr>
<td>$x &gt;= 11$</td>
<td>False</td>
<td>bool</td>
</tr>
</tbody>
</table>
Revisiting our program

```
n = int(input("Enter a positive integer:"))
while n > 0:
    print n % 2
    n = n/2
```

- The boolean expression is **True** when \( n \) is positive and is **False** when \( n \) is less than or equal to 0.

- **Example:** Suppose \( n \) is initially 25. Then \( n \) takes on the values (in this order): 25, 12, 6, 3, 1, 0. When \( n \) becomes 0, the program exits the while-loop.