Understanding our first program

JAN 28TH 2013
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)
- `n = math.sqrt(100)`
  (Can be used after importing the math module)
- `n = raw_input("Enter a number:")`
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 and then the function reads a line from input 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*. 
raw_input returns a string

Try this code snippet. What happens?

```python
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.
Data types in Python

- 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:**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Type a nonnegative integer”</td>
<td>string</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
</tr>
</tbody>
</table>

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

  - integer
  - string
  - floating point
  - boolean
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>
</tr>
</thead>
<tbody>
<tr>
<td>9/2</td>
<td>4</td>
</tr>
<tr>
<td>9.0/2</td>
<td>4.5</td>
</tr>
<tr>
<td>9/2.0</td>
<td>4.5</td>
</tr>
<tr>
<td>5 + 1</td>
<td>6</td>
</tr>
<tr>
<td>5 + 1.0</td>
<td>6.0</td>
</tr>
<tr>
<td>“hello,”+” friend”</td>
<td>“hello, friend”</td>
</tr>
</tbody>
</table>
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(“320”)</td>
<td>320</td>
</tr>
<tr>
<td>float(“320”)</td>
<td>320.0</td>
</tr>
<tr>
<td>str(134)</td>
<td>“134”</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.
How do while statements affect program flow?

Line 1
while boolean expression:
  Line 2
  Line 3
Line 4

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

Is boolean expression true?
  yes
  Line 2
  Line 3
  Line 4

no
Body of while loop

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>$x != 100$</td>
<td>True</td>
</tr>
<tr>
<td>$x &lt;= 10$</td>
<td>True</td>
</tr>
<tr>
<td>$x &gt; -10$</td>
<td>True</td>
</tr>
<tr>
<td>$x &gt;= 11$</td>
<td>False</td>
</tr>
</tbody>
</table>
A silly while loop example

```
 n = int(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.
Another while-loop Example

```python
n = int(input("Enter a positive integer:"))
sum = 0

while n != 0:
    sum = sum + n % 10
    n = n / 10
```

- **What is the output?**
  - If input is 2073?
  - If input is 9999?

- **What does the program do, in general?**