**PYTHON EXAMPLE EXPLANATION**

If2.py

Python is quite unusual in that blocks of statements are defined not with the usual markers, { and }, or begin and end, but by the actual indention of the code. Below is a summary of the rules. Note that blank lines and comments are ignored, so "line" in this discussion means only the non-blank, non-comment lines.

1. The first line starts the first block and must not be indented. Indenting the first line is a syntax error.
2. A line which is indented the same as the one immediately before belongs to the same block.
3. A line which is indented more than the one immediately before it starts a new block.
4. A line *y* which is indented less than the line *x* immediately before it closes *x*'s block and belongs to an earlier block. Specifically:
   1. Line *y* is matched with the most recent line *a* which has the same indent.
   2. If there is no such line *a*, or if some line between *a* and *y* has a lesser indent, *y* is a syntax error. Otherwise:
   3. Line *y* belongs to the same block as *a*.
   4. All blocks started between *a* and *y* (including the one *x* belongs to) are closed.
5. The end-of-file closes all open blocks.

You can find the more official description of all this

**Comparison and Boolean Ops**

Bool.py

Python comparisons are unusual in that chaining comparisons is allowed, and works reasonably. Chained comparisons translate to a series of anded terms, repeating the middle. For instance, a < b == c <= dis interpreted as a < b and b == c and c <= d. The only other note is that the middle expression (4 in the above example) is evaluated only once. For a constant, this doesn't matter, but it will make a difference for expressions which have a side-effect. Note that this does not always have the desired effect. For instance, 3 != 2 != 3 is true, which might not be what you wanted.

Here's how the rest of the world does it:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Expression** |  | **C, C++** |  | **Java, Pascal, Others** |  | **Python** |
| 5 > 4 > 3 |  | Groups as (5 > 4) > 3, which becomes 1 > 3,  which is false. |  | Illegal |  | Translates to 5 > 4 and 4 > 3, which is true. |

Python does have a boolean type, with constants True and False. In addion, the following things are considered false: zero (all types), empty lists, strings, tuples, or dictionaries (the last two of which we have not yet discussed). Also the object None, which is sort of like Java's nil or perl's undef. Comparisons return boolean.

For the boolean operators, and, or and not, we have to break down and write out the words; none of this && and || stuff. And and or are short-circuit. They return the last argument evaluated, which is the one that decided the question. This doesn't make any difference inside an if or while, but it is useful in some contexts. Some other Unix scripting languages operate this way.