CSCI 141
Scott Wehrwein

The bool data type
Boolean Operators
Boolean Expressions
Goals

• Understand the use and values of the type `bool` and the meaning of a boolean expression.

• Understand the behavior of the arithmetic comparison operators: `>`, `<`, `<=`, `>=`, `==`, `!=`

• Understand the behavior of the boolean logical operators `and`, `or`, `not`

• Know where the above operators fit into the order of operations

• Be able to write out a truth table for a boolean expression of two variables.
What have we covered so far?

• Data is stored in memory.
  
  integers are stored using their *binary* representation

• Each piece of data has a type.

  *so far we’ve seen: int, float, str*

• Variables can assign names to pieces of data.

  the assignment operator stores a value in a variable, as in:
  
  `my_var = “Hello, world!”`

• Operators can do things to the data (these operations are performed by the CPU).

  *so far: assignment operator (=)*

  arithmetic operators: (+, -, *, /, **, //, %)
What have we covered so far?

- A function can take inputs (arguments) and can produce an output (return value)
  
  so far: `input`, `print`, `type`, `int`, `float`, `str`

- Statements are instructions that are executed
  
  so far: assignment statements, such as `my_var = 64 + 8`

- Expressions are like phrases that can be evaluated to determine what value they represent.
  
  so far:
  - functions that return values, like `int(42.8)`
  - arithmetic expressions, like `(4 + 2) / 2`
  - and combinations of other expressions, like `(2**3) // int(user_input)`
Some more familiar operators

These ones do what you think.

3 < 4
4 <= 4
6.7 > 6.3
1000 >= 1000

What does 3 < 4 evaluate to?
What does type(3 < 4) evaluate to?
We need a new data type!

\[ a < b \]

can only be one of two things: a **true** statement or a **false** statement.

**Boolean expressions** are expressions that evaluate to one of two possible values: **True** or **False**

What does \( 3 < 4 \) evaluate to? **True**
What does `type(3 < 4)` evaluate to? **`bool`**
The `bool` data type

- Named after 19th century philosopher/mathematician George Boole, who developed Boolean algebra
- A boolean value (`bool`) represents logical propositions that can be either `true` or `false`.
- In Python, these values are reserved keywords: `True` and `False`. Note capitalization.
- Can be used for things like `3 < 4` or `a < b`, but anything else that can be true or false:

  ```python
  is_raining = False
  ```
Comparison Operators

These should be familiar!

<  Less than
>  Greater than
<= Less than or equal to
>= Greater than or equal to
==
!=

Examples:

3 < 4  =>  True
4 <= 4  =>  True
6.7 > 6.3  =>  True
1000 > 1000  =>  False
Comparison Operators

These should be familiar!

<  Less than
>  Greater than
<= Less than or equal to
>= Greater than or equal to
== Equal to
!= Not equal to

Examples:

3 == 4  =>  False
5 != 4  =>  True
4.0 < 4.6  =>  True
Comparison Operators

These should be familiar!

<  Less than
>  Greater than
<= Less than or equal to
>= Greater than or equal to
== Equal to
!= Not equal to

Examples:

3 == 4  =>  False
5 != 4  =>  True
4.0 < 4.6  =>  True

Unlike some operators (e.g., //), the concept of equality has meaning for some non-numeric types:

True == False  =>  False
"abc" == "bcd"  =>  False
"a" == "A"  =>  False
type(4) == type(5)  =>  True
5.0 == 5  =>  True
Logical Operators

<  Less than
>  Greater than
<= Less than or equal to
>= Greater than or equal to
== Equal to
!= Not equal to

a and b is true only when both a and b evaluate to True

a or b is true when at least one of a and b evaluates to True

not switches the value:
not True => False
not False => True

and logical conjunction, logical and
or  logical disjunction, logical or
not logical negation, logical not
Binary vs Unary Operators

- We have already seen some binary operators and one unary operator.

- **Binary operators** take two operands:
  - $a + b$
  - $c \div d$
  - $12 \neq 4$

- **Unary operators** take one operand:
  - $-b$
  - $not \ False$

Notice: minus (-) and plus (+) can behave as unary or binary operators!
Truth Tables for and, or

If $x$ is true and $y$ is true, $x \text{ and } y$ is true.

If $x$ is true and $y$ is false, $x \text{ and } y$ is false.

If $x$ is true and $y$ is true, $x \text{ and } y$ is true.
Truth Tables for **and**, **or**

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
Operator Precedence

- Parentheses
- Exponentiation (right-to-left)
- Multiplication and Division
- Addition and Subtraction

All are evaluated left to right except for exponentiation.
Operator Precedence: Updated

Parentheses

Exponentiation (right-to-left)

Unary + and –

Multiplication and Division

Addition and Subtraction

Numerical comparisons <, >, <=, >=, ==, !=

not

and

or

All are evaluated left to right except for exponentiation.
Operator Precedence: Updated

Parentheses

Exponentiation (right-to-left)

Unary + and –

Multiplication and Division

Addition and Subtraction

Numerical comparisons <, >, <=, >=, ==, !=

not

and

or

Special case:  
\(2**-1 = 0.5\)

Unspecial but surprising case:  
\(-2**2 = -4\)

All are evaluated left to right except for exponentiation.

You can look up all the details: https://docs.python.org/3/reference/expressions.html#operator-precedence
Examples

print(3 != 5 and 4 < 7)

print(3 == 5 or 4 < 7)

print(not False)

print(3 == 5 or 4 > 7)

print(not 6 < 8)
Bigger Example

1 == 6 and True or (1.2 < (5 % 3))
Bigger Example

1 == 6 and True or (1.2 < (5 % 3))

1 == 6 and True or (1.2 < 2)

1 == 6 and True or True

False and True or True

False or True

True