Choices made by a computer provide the appearance of intelligence; billions of choices per second.
Choice is binary:
choose one way or the other.
Choice: Boolean Expressions

- George Boole’s (mid-1800’s) mathematics of logical expressions.
- Boolean expressions (conditions) have a value of True or False.

What is True, and what is False?

Binary:
- 0 is False
  - a zero number or empty object: 0, “”, [ ]
- non-zero is True
  - any nonzero number or nonempty object: 1, 100, “hello”, [a,b]
Also, a special value: “None”; less than everything and equal to nothing.

Every Boolean expression has the form:

\[ expression \text{ operator} \ expression \]

The result is True or False.
Relational Operators

- Less than: <
- Greater than: >
- Equal to: ==  (Not the same as =)
- Not equal to: !=
- Less than or equal to: <=
- Greater than or equal to: =>

Examples

- If my_int = 5, then the value of expression my_int < 7 is True
- If my_char = 'A', then the value of expression my_char == 'Q' is False
Chained comparisons

- In Python, chained comparisons work just like you would expect in a mathematical expression:
- Given `my_int` has the value 5
  - `0 <= my_int <= 5`
  - `True`
  - `0 < my_int <= 5 > 10`
  - `False`

Pitfall: floats are approximations

- Be careful of floating point equality comparisons, especially with zero, e.g. `my_float==0`.
  - Use the converse “!=“ whenever possible.
- `Result = 2/2.0000000000000001`
- `Result == 1.0`
  - `True`
Compound Expressions

- Logically $0 < X < 3$ is actually $(0 < X) \text{ and } (X < 3)$

- Logical Operators (lower case)
  - `and`
  - `or`
  - `not`

Truth Tables

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>q</td>
<td>not p</td>
<td>p and q</td>
<td>p or q</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>-------</td>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Truth Tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Truth Tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

### Truth Tables

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>not p</th>
<th>p and q</th>
<th>p or q</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td></td>
</tr>
</tbody>
</table>

Compound Evaluation

- Logically \( 0 < X < 3 \) is actually
  \((0 < X) \text{ and } (X < 3)\)
- Evaluate using \( X \) with a value of 5:
  \((0 < X) \text{ and } (X < 3)\)
- Parenthesis first: \((\text{True}) \text{ and } (\text{False})\)
- Final value: False

(Note: parenthesis are not necessary in this case.)

Relational operators have precedence and associativity just like numerical operators.

See full list in Appendix F.
Boolean vs. relational operations

- Relational operations (\(<\), \(\rangle\), \(==\), etc.) always return \texttt{True} or \texttt{False}
- Boolean operations (\texttt{and}, \texttt{or}, \texttt{not}) are different in that:
  - They can return values (that represent \texttt{True} or \texttt{False})
  - They have “short circuiting”

Remember!

- \texttt{0}, \texttt{''}, \texttt{[ ]} or other “empty” objects are equivalent to \texttt{False}
- anything else is equivalent to \texttt{True}
Returning values

- \( 2 < 3 \)
- \( 2 \text{ or } 3 \)
- \( 2 \text{ and } 3 \)
- \( \text{True} \)
- \( 2 \)
- \( 3 \)

**Boolean operators** \((\text{and, or, not})\) return the first object (not necessarily \(\text{True} \) or \(\text{False}\)) that establishes the expression value: “short circuit.”

Character Expressions

If \( C = \text{'z'} \)

Values of the following?

- \( C == \text{'z'} \)
- \( C > \text{'a'} \)
  - based on the ASCII char set (Appendix E).
  - Uses an association between each character and an integer. The integer is the value compared (more in the String section).
An Example

- \( A = 3 \)
- \( B = 8 \)
- **Evaluate:** \( A \neq B \) or \( A < -6 \)
- **Precedence:** \((A \neq B)\) or \((A < -6)\)
- **Evaluate:** \((\text{True})\) or \((\text{False})\)
- **Evaluate:** True
- **Shortcut:** final value known after \((A \neq B)\)
Check yourself: try this

1. \(4 > 5.0\)
2. \(x = 2\)
   \(x > 0 \text{ and } x \neq 2\)
3. \(x = 2\)
   \(x > 1 \text{ or } x < 1\)
4. ‘a’ > ‘z’
5. ‘a’ > 2
Selection is how programs make choices, and billions of choices per second provides a lot of the power of computing.
### Sequential Execution

Statement 1

Statement 2

...  

Statement n

### Selective Execution

- **Boolean expression**
  - True: statement 1
  - False: statement 2
Compound statements incorporate a set of statements being used as a group.

Examples: if, for, while

Compound Statement Format

```
keyword expression:
  statement
  statement
  statement
  (optional)
```

- keyword: if, for, while
- expression: logical condition
- statement: code block
- indentation under the keyword
- “suite” of statements
- ends all headers
Python `if` statement, round 1

```python
if Boolean_expression :
suite
```

Evaluate the Boolean Expression (True or False). If True, execute all statements in the suite.

Example #6A

This example illustrates a simple selection statement.
Behavior

- Divide two integers, but check for division by zero before performing the division.
- If division by zero
  - print the error
  - exit the program
- print the result of the division

Warning about indentation

- Elements of the “suite” must all be indented the same number of spaces/tabs
- Python only recognizes suites when they are indented the same “distance”
- You must be careful to get the indentation right to get suites right.
Python Selection (if), Round 2

```python
if Boolean_expression:
suite1
else:
suite2
```

The process is:
- evaluate the Boolean
- if True, run suite1
- if False, run suite2

Example #6B

This example illustrates a simple selection statement.
Python Selection (if), Round 3

```python
if Boolean_expression1:
    suite_1
elif Boolean_expression2:
    suite_2
    (as many elif’s as you want)
else:
    suite_last
```

### if, elif, else, the process

- Evaluate Boolean expressions until:
  - a Boolean expression returns True
  - none of the Boolean expressions return True
- if a Boolean returns True, run the corresponding suite. Skip the rest of the if
- if no Boolean returns True, run the else suite, the default suite
Remember Assignments?

- **Format**: \( \text{lhs} = \text{rhs} \)
- **Behavior**:
  - *expression* in the *rhs* is evaluated producing a value
  - the value produced is placed in the location indicated on the *lhs*
Can do multiple assignments

\[ x, \ y = 2, \ 3 \]  
\# assigns x=2 and y=3

print(x, y)  
\# prints 2 3

Swap

- Initial values: X is 2, Y is 3
- Behavior: swap values of X and Y
  - Note: X=Y; Y=X; doesn’t work
  - introduce extra variable “temp”
    - temp = X  \# save X’s value in temp
    - X=Y  \# assign Y’s value to X
    - Y=temp  \# assign temp’s value to Y
Swap using multiple assignment

\[ x, \ y = 2, \ 3 \]
\[ \text{print} \ (x, \ y) \quad \# \text{ prints } 2 \ 3 \]
\[ x, \ y = y, \ x \]
\[ \text{print} \ (x, \ y) \quad \# \text{ prints } 3 \ 2 \]

Chaining

\[ x = y = 5 \]
\[ \text{print} \ (x, \ y) \quad \# \text{ prints } 5 \ 5 \]
Compound Assignment

\[ y += x \quad \# \text{equivalent to} \quad y = y + x \]

Others

\[ -=, \quad *=, \quad /=, \quad %= \]

Exercise

Use two variables X and Y

1. Use ‘multiple assignment’ to make X = 5 and Y = 11.
2. Increment Y
3. Swap X and Y
4. Use an ‘if’ statement to print the larger value.