CSCI 141

Lecture 7: Conditionals: if, else, elif
Announcements
Announcements

- A2 is out, due next **Tuesday** night. Get started early!
Announcements

• A2 is out, due next **Tuesday** night. Get started early!

• QOTD: late submissions are **not** accepted.
Announcements

• A2 is out, due next **Tuesday** night. Get started early!

• QOTD: late submissions are **not** accepted.
  
  • Not even at 1:07pm, after I've gone over the answers.
Announcements

• A2 is out, due next **Tuesday** night. Get started early!

• QOTD: late submissions are **not** accepted.
  • Not even at 1:07pm, after I've gone over the answers.

• The decimal-to-binary conversion question had an error in the solution. I regraded them all.
Announcements

• A2 is out, due next **Tuesday** night. Get started early!

• QOTD: late submissions are **not** accepted.
  
  - Not even at 1:07pm, after I've gone over the answers.

• The decimal-to-binary conversion question had an error in the solution. I regraded them all.
  
  - Points should be correct, but answers marked "incorrect" may not be incorrect, and vice versa.
Goals

• Understand the behavior of the equality comparison operators (==, !=) on non-numeric types.

• Know how to use an `if` statement to conditionally execute a block of code.

• Know how to use an `if/else` statement to choose which of two code blocks to execute.

• Understand how conditional statements can be nested to make decisions among more than two possibilities.

• Know how to use `if/elif/else` statements.
Last Time

• New type: \texttt{bool}

• New operators:
  • comparison $<$, $>$, $\leq$, $\geq$, $==$, $!=$
  • logical not, and, or

• Operator precedence
Exclusive or ("xor"): True if exactly one of the operands is true. Write the truth table:

<table>
<thead>
<tr>
<th>x</th>
<th>xor</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>F</td>
</tr>
</tbody>
</table>

Note: xor is not a python operator.
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not (a and b) and not(nor a and not b)}
\]

\[
a \text{ and } b \text{ or not (a or b)}
\]

\[
a \text{ or b}
\]

\[
(a \text{ or b) and not (a and b)}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ \text{not} \ (a \ \text{and} \ b) \ \text{and} \ \text{not} \ (\text{not} \ a \ \text{and} \ \text{not} \ b) \]

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } (a \text{ and } b) \text{ and not}(\text{not } a \text{ and } \text{not } b)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not (T and T)} \quad \text{and} \quad \text{not(not T and not T)}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } (T \text{ and } T) \text{ and not } (\text{not } T \text{ and } \text{not } T)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } T \text{ and not (not } T \text{ and not } T)\]

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } T \text{ and not (not } T \text{ and not } T) \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>a</strong></td>
<td><strong>b</strong></td>
</tr>
<tr>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } T \text{ and not } (F \text{ and } F)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to $a \text{ xor } b$?

$$\text{not } T \text{ and not } (F \text{ and } F)$$
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\texttt{not T and not F}

\begin{center}
\begin{tabular}{c|c|c|c|}
 & T & F & \\
\hline
T & & & \\
\hline
F & & & \\
\end{tabular}
\end{center}
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\begin{array}{c}
\text{not } T \\
\text{and } \text{not } F
\end{array}
\]
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ \text{F and not F} \]

<table>
<thead>
<tr>
<th></th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T)</td>
<td>(T)</td>
<td>(T)</td>
</tr>
<tr>
<td>(T)</td>
<td>(F)</td>
<td>(F)</td>
</tr>
<tr>
<td>(F)</td>
<td>(T)</td>
<td>(F)</td>
</tr>
<tr>
<td>(F)</td>
<td>(F)</td>
<td>(T)</td>
</tr>
</tbody>
</table>
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ F \text{ and not } F \]

\[ \begin{array}{c|cc|c}
\text{a} & T & F & b \\
\hline
T & & & F \\
F & & & \\
\end{array} \]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

- F and T
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

F and T
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to \( a \oplus b \)?

\[
\begin{array}{c|c|c|c}
& T & F \\
\hline
T & \text{True} & \text{False} \\
F & \text{False} & \text{True} \\
\end{array}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

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

Answer: F
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } (T \text{ and } F) \text{ and not}(\text{not } T \text{ and not } F)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not} \ (T \ and \ F) \ \text{and} \ \text{not} \ (\text{not} \ T \ and \ \text{not} \ F)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to \( a \ xor \ b \)?

\[
\text{not } F \land \text{not } (F \land T)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

not F and not (F and T)
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } F \land \text{ not } F
\]
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } F \land \text{not } F
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

T and T
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\begin{array}{c|c}
\text{a} & \text{b} \\
T & T \\
T & F \\
F & T \\
F & F \\
\end{array}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

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

Diagram:

- a
  - T
  - F
- b
  - T
  - F

The truth table shows the xor operation:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a xor b</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a \( \text{xor} \) b?

\[
\text{not} \ (F \ \text{and} \ T) \ \text{and} \ \text{not}(\text{not} \ F \ \text{and} \ \text{not} \ T)
\]

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

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to $a \text{ xor } b$?

\[
\text{not } (\text{F and T}) \text{ and not (not F and not T)}
\]

... F/T comes out the same as T/F
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not (F and T) and not(not F and not T)}
\]

... F/T comes out the same as T/F
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

not \((\text{F and F})\) and not(\(\text{not F and not not F}\))
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not (F and F) and not(not F and not F)}
\]

<table>
<thead>
<tr>
<th></th>
<th>F</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\text{not } (F \text{ and } F) \text{ and not} (\text{not } F \text{ and } \text{not } F)
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ \text{not } F \text{ and not } (T \text{ and } T) \]
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ \text{not } F \text{ and not } (T \text{ and } T) \]

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

b

a
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[ \text{not } F \text{ and not } T \]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\begin{array}{c}
\text{not } F \\
\text{and } \text{not } T
\end{array}
\]
QOTD

Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to \( a \ xor \ b \)?

\[
\begin{array}{ccc}
\text{T} & \text{and} & \text{F} \\
\hline
\text{T} & & \text{F} \\
\text{T} & \text{F} & \text{T} \\
\end{array}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

T and F
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

F

\[
\begin{array}{c|c|c}
  & T & F \\
  a & F & T \\
  F & T & T \\
\end{array}
\]
Exclusive or ("xor"): True if exactly one of the operands is true. Which of the following evaluates to a xor b?

\[
\begin{array}{cc}
T & F \\
F & T \\
\end{array}
\]
QOTD

For each expression, give the type and value.

True and False or True

2**3.0

not 1 + 5 // 2 == 3 and 4 < 5 or 4 != 5
Boolean Expressions: Another worked example

What does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
```
Boolean Expressions:
Another worked example

What does this print?

```
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
```

Worked solution is on the following slides for your reference.
Last Time:
Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
```
Last Time:
Boolean Expressions

Another example: what does this print?

\[
\text{print}\left((3 == 5 \text{ or } (3 \neq 5 \text{ and } 5 \neq 7)) \text{ and } 3 < 5\right)
\]
Last Time: Boolean Expressions

Another example: what does this print?

```
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
```
Last Time: Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
```
Last Time:
Boolean Expressions

Another example: what does this print?

```
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or True and True) and 3 < 5)
print((3 == 5 or True) and 3 < 5)
```
Last Time: Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or True and True) and 3 < 5)
print((3 == 5 or True) and 3 < 5)
```
Last Time:
Boolean Expressions

Another example: what does this print?

```
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or ( True and True )) and 3 < 5)
print(( False or True ) and 3 < 5)
print((3 == 5 or True ) and 3 < 5)
print(( False or True ) and 3 < 5)
```
Last Time: Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
print((False or True) and 3 < 5)
print((False or True) and 3 < 5)
```
Last Time:
Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
print((False or True) and 3 < 5)
print((False or True) and 3 < 5)
print(True and 3 < 5)
```
Last Time: Boolean Expressions

Another example: what does this print?

print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
print((False or True) and 3 < 5)
print(True and 3 < 5)
Last Time: Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
print((3 == 5 or True) and 3 < 5)
print((False or True) and 3 < 5)
print(True and 3 < 5)
print(True and True)
```
Last Time:
Boolean Expressions

Another example: what does this print?

```python
print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or (True and True)) and 3 < 5)
print((3 == 5 or True) and 3 < 5)
print((False or True) and 3 < 5)
print(True and 3 < 5)
print(True and True)
```
Last Time:
Boolean Expressions

Another example: what does this print?

print((3 == 5 or (3 != 5 and 5 != 7)) and 3 < 5)
print((3 == 5 or ( True and True )) and 3 < 5)
print((3 == 5 or True ) and 3 < 5)
print(( False or True ) and 3 < 5)
print( True and 3 < 5)
print( True and True )
print( True )
Operator Precedence, Updated Again

- **Parentheses**
- **Exponentiation (right-to-left)**
- **Unary + and -**
- **Addition and Subtraction**
- **Multiplication and Division**
- **Numerical comparisons**: `<`, `>`, `<=`, `>=`, `==`, `!=`
- **Not**
- **And**
- **Or**

All evaluated left to right except for exponentiation.

You can look up all the details: [https://docs.python.org/3/reference/expressions.html#operator-precedence](https://docs.python.org/3/reference/expressions.html#operator-precedence)
Operator Precedence, Updated Again

- 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.

You can look up all the details: https://docs.python.org/3/reference/expressions.html#operator-precedence
Operator Precedence, Updated Again

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.

You can look up all the details: https://docs.python.org/3/reference/expressions.html#operator-precedence
Operator Precedence, Updated Again

Parentheses

Exponentiation (right-to-left)

Unary + and –

Multiplication and Division

Addition and Subtraction

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

not

and

or

Special case: 2**-1 = 0.5

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
Operator Precedence, Updated Again

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
Equality Comparisons

• The operators == and != check whether two values are equal or not.

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

```
4 == 5
"abc" == "bcd"
"abc" == "abc"
type(4) == type(5)
5.0 == 5
```
Equality Comparisons

- The operators `==` and `!=` check whether two values are equal or not.

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

```
4 == 5               => False
"abc" == "bcd"
"abc" == "abc"
type(4) == type(5)
5.0 == 5
```
Equality Comparisons

• The operators == and != check whether two values are equal or not.

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

\[
\begin{align*}
4 & == 5 \quad \Rightarrow \text{False} \\
"abc" & == "bcd" \quad \Rightarrow \text{False} \\
"abc" & == "abc" \\
type(4) & == type(5) \\
5.0 & == 5
\end{align*}
\]
Equality Comparisons

• The operators == and != check whether two values are equal or not.

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

```python
4 == 5 => False
"abc" == "bcd" => False
"abc" == "abc" => True
type(4) == type(5)
5.0 == 5
```
Equality Comparisons

• The operators == and != check whether two values are equal or not.

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

- $4 == 5$ => False
- "abc" == "bcd" => False
- "abc" == "abc" => True
- `type(4) == type(5)` => True
- 5.0 == 5
Equality Comparisons

• The operators == and != check whether two values are equal or not.

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

```
4 == 5 => False
"abc" == "bcd" => False
"abc" == "abc" => True
```

```
type(4) == type(5) => True
5.0 == 5 => True
```
Equality Comparisons

Lightning round!

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

"Scott" == "scott"
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)
Equality Comparisons

Lightning round!

10 == 4 + 6

"abc" == "ab" + "c"

'abc' == "abc"

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)

int(5.6) != int(5.1)

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c"

'abc' == "abc"

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)

int(5.6) != int(5.1)
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c" => True

'abc' == "abc"

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)

int(5.6) != int(5.1)
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c" => True

'abc' == "abc" => True

"Scott" == "scott"

(4+3 > 5) == (1.0 > 4)

int(5.6) != int(5.1)
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c" => True

'abc' == "abc" => True

"Scott" == "scott" => False

(4+3 > 5) == (1.0 > 4)

int(5.6) != int(5.1)
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c" => True

'abc' == "abc" => True

"Scott" == "scott" => False

(4+3 > 5) == (1.0 > 4) => False

int(5.6) != int(5.1) => False

True or False?
Equality Comparisons

Lightning round!

10 == 4 + 6 => True

"abc" == "ab" + "c" => True

'abc' == "abc" => True

"Scott" == "scott" => False

(4+3 > 5) == (1.0 > 4) => False

int(5.6) != int(5.1) => False
Today
Today

• Last week: everything you already knew how to do using a calculator.
Today

• Last week: everything you already knew how to do using a calculator.
Today

- Last week: everything you already knew how to do using a calculator.

- Last lecture: representing and manipulating boolean (true/false) expressions and values.
Today
Today

• Last week: everything you already knew how to do using a calculator.

• Last lecture: representing and manipulating boolean (true/false) expressions and values.
Today

• Last week: everything you already knew how to do using a calculator.

• Last lecture: representing and manipulating boolean (true/false) expressions and values.

• Today: Making **decisions** based on the value of a boolean expression.
Today

• Last week: everything you already knew how to do using a calculator.

• Last lecture: representing and manipulating boolean (true/false) expressions and values.

• Today: Making **decisions** based on the value of a boolean expression.
Today

- Last week: everything you already knew how to do using a calculator.
- Last lecture: representing and manipulating boolean (true/false) expressions and values.
- Today: Making **decisions** based on the value of a boolean expression.
- Also: a new kind of **statement**!
Let’s talk about the weather
Let’s talk about the weather

• You wish to write a software system that recommends what to wear/bring based on the current weather conditions.
Let’s talk about the weather

• You wish to write a software system that recommends what to wear/bring based on the current weather conditions.

• In a later version, you will hook your software up to automated weather sensors that read temperature, wind, and precipitation data in real time.
Let’s talk about the weather

• You wish to write a software system that recommends what to wear/bring based on the current weather conditions.

• In a later version, you will hook your software up to automated weather sensors that read temperature, wind, and precipitation data in real time.

• For now, we’ll just ask the user.
Let’s talk about the weather

Suppose we have bool variable is_raining

Here’s the logic (pseudocode):

• if it is raining, tell the user to bring a raincoat

Here’s the Python code:
Let’s talk about the weather

Suppose we have bool variable is_raining

Here’s the logic (pseudocode):

• if it is raining, tell the user to bring a raincoat

Here’s the Python code:

```python
if is_raining:
    print("You should wear a raincoat!")
```
Let’s talk about the weather

Suppose we have bool variables is_raining and is_windy

Here’s the logic (pseudocode):

- if it is raining and windy, tell the user to bring a raincoat
- if it is raining and not windy, tell the user to bring an umbrella

Here’s the Python code:
Let’s talk about the weather

Suppose we have bool variables is_raining and is_windy.

Here’s the logic (pseudocode):

• if it is raining and windy, tell the user to bring a raincoat.
• if it is raining and not windy, tell the user to bring an umbrella.

Here’s the Python code:

```python
if is_raining and is_windy:
    print("You should wear a raincoat!")
if is_raining and not is_windy:
    print("You should bring an umbrella")
```
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!"")
```
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```
The **if** statement

```
if is_raining:
    print("You should wear a raincoat!")
```
The *if* statement

- **if** keyword
- A boolean expression (the *condition*)
- A colon :

```
if is_raining:
    print("You should wear a raincoat!")
```

An indented **code block**: one or more statements to be executed if the boolean expression evaluates to **True**
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```

**Notes:**
- **if keyword**
- A boolean expression (the condition)
- A colon:
- An indented code block: one or more statements to be executed if the boolean expression evaluates to **True**
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```

**Notes:**

- In Python, the indentation is **required**.
The if statement

if keyword a boolean expression (the condition) a colon:

```python
if is_raining:
    print("You should wear a raincoat!")
```

an indented code block: one or more statements to be executed if the boolean expression evaluates to True

Notes:
- In Python, the indentation is required.
- Indenting with tabs or spaces is acceptable.
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```

**Notes:**
- In Python, the indentation is **required**.
- Indenting with tabs or spaces is acceptable.
- We’ll use the most common convention: indent 4 spaces beyond the line with the `if`
The **if** statement

```python
if is_raining:
    print("You should wear a raincoat!")
```

**Notes:**
- In Python, the indentation is **required**.
- Indenting with tabs or spaces is acceptable.
- We’ll use the most common convention: indent 4 spaces beyond the line with the `if`
- Thonny follows this convention for you.
Demo
Demo

- using the is_raining example
- if statement with a condition that evaluates to True vs False
- statements after the indented code block
- multiple lines in the code block
What if it’s not raining?

What if we want to also print something in case it’s not raining?

```python
if is_raining:
    print("Wear a raincoat!")
```
What if it’s not raining?

What if we want to also print something in case it’s not raining?

```python
if is_raining:
    print("Wear a raincoat!")
if not is_raining:
    print("Don’t wear a raincoat!")
```
What if it’s not raining?

What if we want to also print something in case it’s not raining?

```python
if is_raining:
    print("Wear a raincoat!")
if not is_raining:
    print("Don’t wear a raincoat!")
```

How many times did we check the value of `is_raining`?
What if it’s not raining?

What if we want to also print something in case it’s not raining?

```python
if is_raining:
    print("Wear a raincoat!")
if not is_raining:
    print("Don’t wear a raincoat!")
```

How many times did we check the value of `is_raining`?

Could we do any better?
What if it’s not raining?

What if we want to also print something in case it’s not raining?

```python
if is_raining:
    print("Wear a raincoat!")
if not is_raining:
    print("Don’t wear a raincoat!")
```

How many times did we check the value of `is_raining`?

Could we do any better?

Yes: it’s a common use case to want to choose between two paths of execution (two code blocks).
The *if*/*else* Statement

```python
if isRaining:
    print("Wear a raincoat!")
else:
    print("Don’t wear a raincoat!")
```
The if/else Statement

- **if** keyword
- a boolean expression (the condition)

```python
if isRaining:
    print("Wear a raincoat!")
else:
    print("Don’t wear a raincoat!")
```

*an indented code block to be executed if the condition evaluates to True*
```python
if isRaining:
    print("Wear a raincoat!")
else:
    print("Don’t wear a raincoat!")
```
The `if/else` Statement

- **if keyword**
- a boolean expression (the **condition**)
- an indented code block to be executed if the condition evaluates to **True**

```
if isRaining:
    print("Wear a raincoat!")
else:
    print("Don’t wear a raincoat!")
```

- **else keyword**
- an indented code block to be executed if the condition evaluates to **False**
- a colon:
What does the following program print?

```python
if 2 + 5 == 5:
    print(2 + 5)
else:
    print("not equal")
```

A. 2 + 5
B. 7
C. 2 + 5 == 5
D. not equal
if/else: the basics

What does the following program print?

```python
a = 5
if a >= 5 and a <= 5:
    print(a)
else:
    print("nope")
```

A. 5  
B. a >= 5  
C. a <= 5  
D. nope
if/else: the basics

What does the following program print?

```python
a = 5
if a >= 5 and a <= 5:
    print(a)
else:
    print("nope")
```

A. 5  
B. a >= 5  
C. a <= 5  
D. nope

Is there a better way to write the condition?
Aim for Simplicity

\[
a = 5 \\
\text{if } a \geq 5 \text{ and } a \leq 5: \\
\quad \text{print}(a) \\
\text{else:} \\
\quad \text{print}(\text{"nope"})
\]

\[
a = 5 \\
\text{if } a = 5: \\
\quad \text{print}(a) \\
\text{else:} \\
\quad \text{print}(\text{"nope"})
\]
Aim for Simplicity

The program on the right does exactly the same thing, but is easier to read, and therefore is preferable.
Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?
Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?

```python
# assume x and y are numbers
if x < y:
    if x < y:
        print("x is less than y")
```
Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?

```python
# assume x and y are numbers
if x < y:
    print("x is less than y")
else:
```

Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?

```python
# assume x and y are numbers
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```
Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?

```python
# assume x and y are numbers
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```

the inner if/else statement is the indented code block for the else clause of the outer if/else statement.
Nested Conditionals

If/else lets you choose between two options.

What if there are more than two possibilities?

```python
# assume x and y are numbers
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```

Note: the conditions still have to be boolean expressions (i.e., they evaluate to True or False)

the inner if/else statement is the indented code block for the else clause of the outer if/else statement.
Nested Conditionals

Suppose $x = 4$ and $y = 5$. How many comparison operators ($<$, $>$) are evaluated by the following code?

```python
# assume x and y are numbers
if x < y:
    print("x is less than y")
else:
    if x > y:
        print("x is greater than y")
    else:
        print("x and y must be equal")
```

A. 0  
B. 1  
C. 2  
D. 3
Chained Conditionals: Demo

**Task:** Write a program to ask the user for their 141 section number and print out when their lab section happens.

```python
>>> %Run section_times.py
    Enter your CSCI 141 section number: 40372
    Your lab is on Monday from 10 - 12.
```

```bash
>>> 
```
Chained Conditionals: Demo
Chained Conditionals: Demo

- sections.py: with chained if/else statements
- sections_elif.py: with if/elif/else
- sections_refactored.py: refactored to set variables then call print once
- sections_refactored.py: with feature to check for conflicts with lab
if isRaining and not isWindy:
    print("Bring an umbrella!"")
elif isRaining and isWindy:
    print("Wear a raincoat!")
else:
    print("No rain gear needed!")