Lecture 4: Boolean Expressions

Comp 102

Forman Christian University

Recap

Pop Quiz: What type is it?

1. `42`
2. `3.14`
3. `"hello"`
4. `True`
5. `None`

Pop Quiz: What type is it?

1. `42` — `int`
2. `3.14`
3. `"hello"`
4. `True`
5. `None`

Pop Quiz: What type is it?

1. `42` — `int`
2. `3.14` — `float`
3. `"hello"`
4. `True`
5. `None`

Pop Quiz: What type is it?

1. `42` — `int`
2. `3.14` — `float`
3. `"hello"` — `str`
4. `True`
5. `None`

Pop Quiz: What type is it?

1. `42` — `int`
2. `3.14` — `float`
3. `"hello"` — `str`
4. `True` — `bool`
5. `None`

Pop Quiz: What type is it?

1. `42` — `int`
2. `3.14` — `float`
3. `"hello"` — `str`
4. `True` — `bool`
5. `None` — `NoneType`

Pop Quiz: Evaluate

Let `x = 5` and `y = 2`

1. `x + y`
2. `x ** y`
3. `x / y`
4. `x // y`

Pop Quiz: Evaluate

Let `x = 5` and `y = 2`

1. `x + y` 7
2. `x ** y`
3. `x / y`
4. `x // y`

Pop Quiz: Evaluate

Let `x = 5` and `y = 2`

1. `x + y` — 7
2. `x ** y` — 25
3. `x / y`
4. `x // y`

Pop Quiz: Evaluate

Let `x = 5` and `y = 2`

1. `x + y` — 7
2. `x ** y` — 25
3. `x / y` — 2.5
4. `x // y`

Pop Quiz: Evaluate

Let `x = 5` and `y = 2`

1. `x + y` — 7
2. `x ** y` — 25
3. `x / y` — 2.5
4. `x // y` — 2

The Boolean Type

George Boole

- English mathematician
- Invented **Boolean algebra**
- Logic with only two values:

George Boole

- English mathematician
- Invented **Boolean algebra**
- Logic with only two values:

`True` `False`

`bool` in Python

```
>>> True
True
```

`bool` in Python

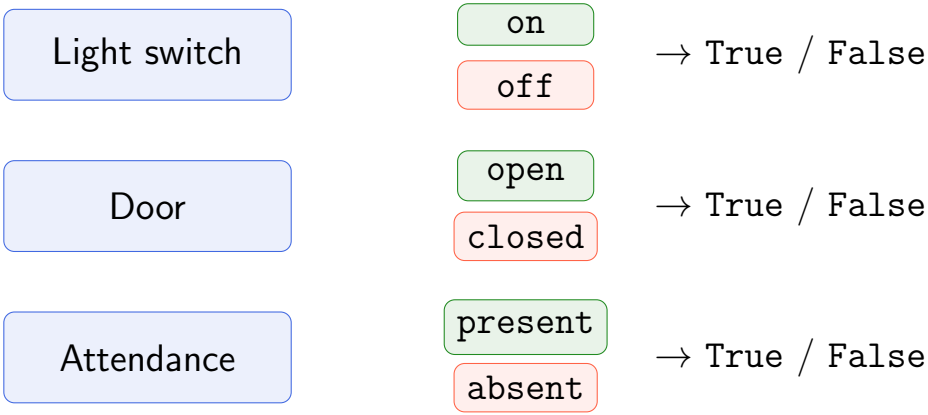
```
>>> True
True
>>> type(True)
<class 'bool'>
```

`bool` in Python

```
>>> True
True
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Recall: `bool` is a **scalar** type, like `int`, `float`, `NoneType`

Booleans are Everywhere



Careful: Case Matters!

Python is **case-sensitive**. Only these are valid:

`True` ✓

`False` ✓

Careful: Case Matters!

Python is **case-sensitive**. Only these are valid:

`True`	✓	Valid boolean
`False`	✓	Valid boolean

Careful: Case Matters!

Python is **case-sensitive**. Only these are valid:

`True`	✓	Valid boolean
`False`	✓	Valid boolean
`true`	✗	`NameError`
`FALSE`	✗	`NameError`
`TRUE`	✗	`NameError`

Comparison Operators

Boolean Expressions:

- evaluate to either **True** or **False**

Comparison:

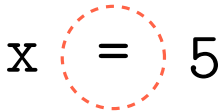
Math	CS
`x = 5`	`x == 5`

Boolean Expressions:

- evaluate to either **True** or **False**

Comparison:

Math



CS

Assignment vs Comparison

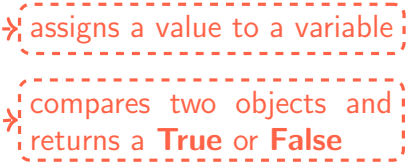
```
>>> a = 5
>>> b = 3
>>> (a + b) == 8
```

assigns a value to a variable

Assignment vs Comparison

```
>>> a = 5
>>> b = 3
>>> (a + b) == 8
```

True



Assignment vs Comparison

```
>>> a = 5
>>> b = 3
>>> (a + b) == 8
```

assigns a value to a variable

compares two objects and returns a **True** or **False**

True

Syntax:

$$\underbrace{< \texttt{some\_exp} > == < \texttt{other\_exp} >}$$

This entire line will be replaced with **True** *or* **False**

Comparison Operators

Compares two objects and returns a **True** or **False** value

- `i < j` → less than
- `i <= j` → less than or equal to
- `i > j` → greater than
- `i >= j` → greater than or equal to
- `i == j` → equality test, **True** if *i* and *j* are the same
- `i != j` → inequality test, **True** if *i* and *j* are not the same

Equality: == and !=

```
>>> 5 == 5
True
```

Equality: == and !=

```
>>> 5 == 5
True
>>> 5 == 6
False
```

`==` returns **True** when both sides are **equal**

Equality: == and !=

```
>>> 5 == 5
True
>>> 5 == 6
False
>>> 5 != 6
True
```

`==` returns **True** when both sides are **equal**

Equality: == and !=

```
>>> 5 == 5
True
>>> 5 == 6
False
>>> 5 != 6
True
>>> 5 != 5
False
```

== returns **True** when both sides are **equal**

!= returns **True** when both sides are **not equal**

Ordering: < > <= >=

```
>>> 3 < 5
True
```

Ordering: < > <= >=

```
>>> 3 < 5
True
>>> 5 < 3
False
```

Ordering: < > <= >=

```
>>> 3 < 5
True
>>> 5 < 3
False
>>> 5 <= 5
True
```

Ordering: < > <= >=

```
>>> 3 < 5
True
>>> 5 < 3
False
>>> 5 <= 5
True
```

Watch out for <= and >=

5 <= 5 is **True** because
5 is *equal to* 5

The "**or equal to**" part matters!

Comparing Expressions

You can compare **any** expressions, not just simple values:

```
>>> (2 + 3) == 5
True
```

Comparing Expressions

You can compare **any** expressions, not just simple values:

```
>>> (2 + 3) == 5
True
>>> len("hello") > 3
True
```

Each side is evaluated first, then compared.

Comparing Expressions

You can compare **any** expressions, not just simple values:

```
>>> (2 + 3) == 5
True
>>> len("hello") > 3
True
>>> max(4, 7) == min(7, 9)
True
```

Each side is evaluated first, then compared.

You Try

Type the following in the Python shell. Let:

```
x = 2
y = 3
```

1. `x > y`
2. `(x + 1) == max(x, y)`
3. `x > x`
4. `x != (y - 1)`

You Try

Type the following in the Python shell. Let:

```
x = 2
y = 3
```

1. `x > y` — False
2. `(x + 1) == max(x, y)`
3. `x > x`
4. `x != (y - 1)`

You Try

Type the following in the Python shell. Let:

```
x = 2
y = 3
```

1. `x > y` — False
2. `(x + 1) == max(x, y)` — True
3. `x > x`
4. `x != (y - 1)`

You Try

Type the following in the Python shell. Let:

```
x = 2
y = 3
```

1. `x > y` — False
2. `(x + 1) == max(x, y)` — True
3. `x > x` — False
4. `x != (y - 1)`

You Try

Type the following in the Python shell. Let:

```
x = 2
y = 3
```

1. `x > y` — False
2. `(x + 1) == max(x, y)` — True
3. `x > x` — False
4. `x != (y - 1)` — False

You Try

Now let:

```
a = 10
b = 3
```

1. `a >= 10`
2. `a == (b * 3)`
3. `(a // b) == 3`
4. `abs(b - a) > 5`

You Try

Now let:

```
a = 10
b = 3
```

1. `a >= 10` — `True`
2. `a == (b * 3)`
3. `(a // b) == 3`
4. `abs(b - a) > 5`

You Try

Now let:

```
a = 10
b = 3
```

1. `a >= 10` — True
2. `a == (b * 3)` — False
3. `(a // b) == 3`
4. `abs(b - a) > 5`

You Try

Now let:

```
a = 10
b = 3
```

1. `a >= 10` — True
2. `a == (b * 3)` — False
3. `(a // b) == 3` — True
4. `abs(b - a) > 5`

You Try

Now let:

```
a = 10
b = 3
```

1. `a >= 10` — True
2. `a == (b * 3)` — False
3. `(a // b) == 3` — True
4. `abs(b - a) > 5` — True

Logical Operators

The `not` Operator

Flips a boolean value: `True` ↔ `False`

A	not A
`True`	
`False`	

The `not` Operator

Flips a boolean value: `True` $\leftrightarrow$ `False`

A	not A
`True`	`False`
`False`	`True`

The `not` Operator

Flips a boolean value: `True` ↔ `False`

A	not A
True	False
False	True

```
>>> not True
False
>>> not (5 > 3)
False
```

`not` Step by Step

```
not (5 > 3)
```

`not` Step by Step

```
not (5 > 3)
```

Step 1: Evaluate inside the parentheses

`5 > 3` → `True`

`not` Step by Step

```
not (5 > 3)
```

Step 1: Evaluate inside the parentheses

`5 > 3` → `True`

Step 2: Apply `not`

`not True` → `False`

The `and` Operator

`True` **only** when **both** are `True`

"I'll go out if it's sunny **AND** *I'm free"*

A	B	A and B
True	True	
True	False	
False	True	
False	False	

The `and` Operator

`True` **only** when **both** are `True`

"I'll go out if it's sunny **AND** *I'm free"*

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

The `and` Operator

`True` **only** when **both** are `True`

"I'll go out if it's sunny **AND** *I'm free"*

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False

```
>>> True and True
True
>>> True and False
False
```

and Step by Step

```
(5 > 3) and (2 > 4)
```

`and` Step by Step

```
(5 > 3) and (2 > 4)
```

Step 1: Evaluate left side

`5 > 3` → `True`

Step 2: Evaluate right side

`2 > 4` → `False`

`and` Step by Step

```
(5 > 3) and (2 > 4)
```

Step 1: Evaluate left side

`5 > 3` → `True`

Step 2: Evaluate right side

`2 > 4` → `False`

Step 3: Combine with `and`

`True and False` → `False`

The `or` Operator

`True` when **at least one** is `True`

"I'll be happy if I get pizza **OR** *ice cream"*

A	B	A or B
True	True	
True	False	
False	True	
False	False	

The `or` Operator

`True` when **at least one** is `True`

"I'll be happy if I get pizza **OR** *ice cream"*

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

The `or` Operator

`True` when **at least one** is `True`

"I'll be happy if I get pizza **OR** *ice cream"*

A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False

```
>>> False or True
True
>>> False or False
False
```

`or` Step by Step

```
(10 < 5) or (3 == 3)
```

`or` Step by Step

```
(10 < 5) or (3 == 3)
```

Step 1: Evaluate left side

`10 < 5` → `False`

Step 2: Evaluate right side

`3 == 3` → `True`

`or` Step by Step

```
(10 < 5) or (3 == 3)
```

Step 1: Evaluate left side

`10 < 5` → `False`

Step 2: Evaluate right side

`3 == 3` → `True`

Step 3: Combine with `or`

`False or True` → `True`

Logical Operators on Bool

a and b are variable names *(with boolean values)*

`not` a → `True` if a is `False`; `False` if a is `True`

a `and` b → `True` if both are `True`

a `or` b → `True` if either or both are `True`

A	B	A and B	A or B
True	True		
True	False		
False	True		
False	False		

Logical Operators on Bool

a and b are variable names *(with boolean values)*

`not` a → `True` if a is `False`; `False` if a is `True`

a `and` b → `True` if both are `True`

a `or` b → `True` if either or both are `True`

A	B	A and B	A or B
True	True	True	
True	False	False	
False	True	False	
False	False	False	

Logical Operators on Bool

a and b are variable names *(with boolean values)*

`not` a → `True` if a is `False`; `False` if a is `True`

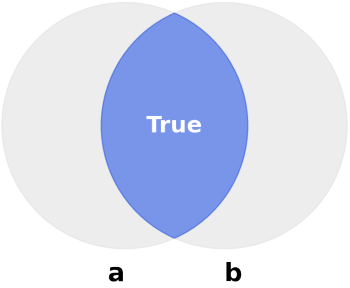
a `and` b → `True` if both are `True`

a `or` b → `True` if either or both are `True`

A	B	A and B	A or B
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

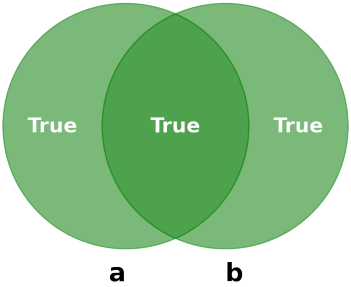
Remember

a and b



`and` is **strict — Both!**

a or b



`or` is **generous — Either!**

You Try

Evaluate each expression:

1. `not (10 > 5)`
2. `(3 > 1) and (2 > 4)`
3. `(3 > 1) or (2 > 4)`
4. `not False and True`

You Try

Evaluate each expression:

1. `not (10 > 5)` — `False`
2. `(3 > 1) and (2 > 4)`
3. `(3 > 1) or (2 > 4)`
4. `not False and True`

You Try

Evaluate each expression:

1. `not (10 > 5)` — False
2. `(3 > 1) and (2 > 4)` — False
3. `(3 > 1) or (2 > 4)` — True
4. `not False and True`

You Try

Evaluate each expression:

1. `not (10 > 5)` — False
2. `(3 > 1) and (2 > 4)` — False
3. `(3 > 1) or (2 > 4)` — True
4. `not False and True` — True

You Try

Evaluate each expression:

1. `not (10 > 5)` — `False`
2. `(3 > 1) and (2 > 4)` — `False`
3. `(3 > 1) or (2 > 4)` — `True`
4. `not False and True` — `True`

Hint for #4: `not` applies to `False` first → `True and True`

`not` Binds First

`not` applies to the **next value only**, before `and`/`or`

Example 1: `not False and True`

`not` Binds First

`not` applies to the **next value only**, before `and`/`or`

Example 1: `not False and True`

Step 1: `not False` → `True`

Step 2: `True and True` → `True`

`not` Binds First

`not` applies to the **next value only**, before `and`/`or`

Example 1: `not False and True`
- Step 1: `not False` → `True`
- Step 2: `True and True` → `True`

Example 2: `not True or False`
- Step 1: `not True` → `False`
- Step 2: `False or False` → `False`

`not` Binds First

`not` applies to the **next value only**, before `and`/`or`

Example 1: `not False and True`
- Step 1: `not False` → `True`
- Step 2: `True and True` → `True`

Example 2: `not True or False`
- Step 1: `not True` → `False`
- Step 2: `False or False` → `False`

Tip: Use parentheses to be explicit!

`(not True) or False` vs `not (True or False)`

Compound Boolean Expressions

Is x between 1 and 10?

```
x = 5
```

Is x between 1 and 10?

```
x = 5
```

Expression: `x >= 1 and x <= 10`

Is x between 1 and 10?

```
x = 5
```

Expression: `x >= 1 and x <= 10`

Step 1: `5 >= 1` → `True`

Step 2: `5 <= 10` → `True`

Step 3: `True and True` → `True`

x is in range!

Is x outside 1 to 10?

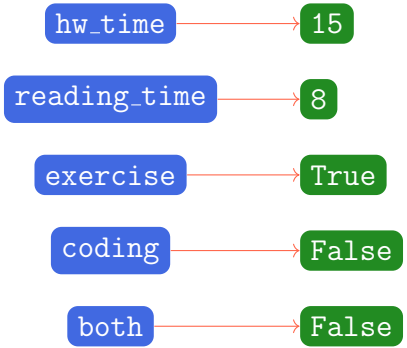
Is x outside 1 to 10?

Expression: `x < 1 or x > 10`

Use `or` because x can't be in **both** regions at once

Comparison Example

```
hw_time = 15
reading_time = 8
print(reading_time > hw_time)
exercise = True
coding = False
both = exercise and coding
print(both)
```



You Try

Let `age = 20` and `gpa = 3.5`

Translate three English phrases into Python boolean expressions:

1. "age is between 18 and 65"
2. "gpa is above 3.0 or age is under 25"
3. "age is NOT under 18"

You Try

Let `age = 20` and `gpa = 3.5`

Translate three English phrases into Python boolean expressions:

1. "age is between 18 and 65"
   `age >= 18 and age <= 65` → `True`
2. "gpa is above 3.0 or age is under 25"
3. "age is NOT under 18"

You Try

Let `age = 20` and `gpa = 3.5`

Translate three English phrases into Python boolean expressions:

1. "age is between 18 and 65"
   `age >= 18 and age <= 65` → `True`
2. "gpa is above 3.0 or age is under 25"
   `gpa > 3.0 or age < 25` → `True`
3. "age is NOT under 18"

You Try

Let `age = 20` and `gpa = 3.5`

Translate three English phrases into Python boolean expressions:

1. "age is between 18 and 65"
   `age >= 18 and age <= 65` → `True`
2. "gpa is above 3.0 or age is under 25"
   `gpa > 3.0 or age < 25` → `True`
3. "age is NOT under 18"
   `not (age < 18)` → `True`

Operator Precedence

Order of Operations

Python evaluates in this order (highest priority first):

Priority	Operators	Example
1 (highest)	`**`	`2 ** 3`
2	`* / // %`	`6 / 2`
3	`+ -`	`3 + 4`
4	`< <= > >= == !=`	`x > 5`
5	`not`	`not True`
6	`and`	`a and b`
7 (lowest)	`or`	`a or b`

Arithmetic → Comparison → `not` → `and` → `or`

Precedence in Action

```
3 + 2 > 4 and not False
```

Precedence in Action

```
3 + 2 > 4 and not False
```

Step 1 (arithmetic): `3 + 2 > 4 and not False`

$\rightarrow$ `5 > 4 and not False`

Precedence in Action

`3 + 2 > 4 and not False`

Step 1 (arithmetic): `3 + 2 > 4 and not False`

$\rightarrow$ `5 > 4 and not False`

Step 2 (comparison): `5 > 4 and not False`

$\rightarrow$ `True and not False`

Precedence in Action

`3 + 2 > 4 and not False`

Step 1 (arithmetic): `3 + 2 > 4 and not False`

$\rightarrow$ `5 > 4 and not False`

Step 2 (comparison): `5 > 4 and not False`

$\rightarrow$ `True and not False`

Step 3 (`not`): `True and not False`

$\rightarrow$ `True and True`

Precedence in Action

`3 + 2 > 4 and not False`

Step 1 (arithmetic): `3 + 2 > 4 and not False`

$\rightarrow$ `5 > 4 and not False`

Step 2 (comparison): `5 > 4 and not False`

$\rightarrow$ `True and not False`

Step 3 (`not`): `True and not False`

$\rightarrow$ `True and True`

Step 4 (`and`): `True and True` $\rightarrow$ `True`

Tricky!

```
True or False and False
```

Tricky!

`True or False and False`

`and` binds tighter than `or`, so:

→ `True or (False and False)`
→ `True or False`
→ `True`

Tricky!

`True or False and False`

`and` binds tighter than `or`, so:

→ `True or (False and False)`
→ `True or False`
→ `True`

Big Idea: When in doubt, use **parentheses**! They make your code readable and remove ambiguity.

You Try

Add parentheses to show the order, then evaluate:

1. `not True or True`
2. `5 > 3 and 2 + 1 == 3`
3. `not 4 > 5 or 3 < 2`

You Try

Add parentheses to show the order, then evaluate:

1. `not True or True`
   `(not True) or True` → `False or True` → `True`
2. `5 > 3 and 2 + 1 == 3`
3. `not 4 > 5 or 3 < 2`

You Try

Add parentheses to show the order, then evaluate:

1. `not True or True`
   `(not True) or True` → `False or True` → `True`
2. `5 > 3 and 2 + 1 == 3`
   `(5 > 3) and ((2 + 1) == 3)` → `True and True` → `True`
3. `not 4 > 5 or 3 < 2`

You Try

Add parentheses to show the order, then evaluate:

1. `not True or True`
   `(not True) or True` → `False or True` → `True`
2. `5 > 3 and 2 + 1 == 3`
   `(5 > 3) and ((2 + 1) == 3)` → `True and True` → `True`
3. `not 4 > 5 or 3 < 2`
   `(not (4 > 5)) or (3 < 2)` → `True or False` → `True`

Truthy and Falsy

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
```

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
True
```

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
True
```

```
>>> bool(0)
False
```

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(-5)
True
```

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(-5)
True
>>> bool(0.0)
False
```

Type Casting: `bool()`

Just like `int()`, `float()`, `str()` — we can cast to `bool`:

```
>>> bool(1)
True
>>> bool(0)
False
>>> bool(-5)
True
>>> bool(0.0)
False
```

Rule: For numbers, **zero is False**, everything else is **True**

`bool()` on Strings and None

```
>>> bool("hello")
```

`bool()` on Strings and None

```
>>> bool("hello")
True
```

`bool()` on Strings and None

```
>>> bool("hello")
True
```

```
>>> bool("")
False
```

`bool()` on Strings and None

```
>>> bool("hello")
True
```

```
>>> bool("")
False
```

```
>>> bool(None)
False
```

`bool()` on Strings and None

```
>>> bool("hello")
True
```

```
>>> bool("")
False
```

```
>>> bool(None)
False
```

Summary of "Falsy" values:

Falsy (= False)	Truthy (= True)
0, 0.0	Any non-zero number
"" (empty string)	Any non-empty string
None	Almost everything else
False	True

You Try

What does `bool()` return for each?

1. `bool(42)`
2. `bool(0)`
3. `bool("False")`
4. `bool("")`
5. `bool(" ")`

You Try

What does `bool()` return for each?

1. `bool(42)` — `True`
2. `bool(0)`
3. `bool("False")`
4. `bool("")`
5. `bool(" ")`

You Try

What does `bool()` return for each?

1. `bool(42)` — `True`
2. `bool(0)` — `False`
3. `bool("False")`
4. `bool("")`
5. `bool(" ")`

You Try

What does `bool()` return for each?

1. `bool(42)` — `True`
2. `bool(0)` — `False`
3. `bool("False")` — `True` (non-empty string!)
4. `bool("")`
5. `bool(" ")`

You Try

What does `bool()` return for each?

1. `bool(42)` — `True`
2. `bool(0)` — `False`
3. `bool("False")` — `True` (non-empty string!)
4. `bool("")` — `False`
5. `bool(" ")`

You Try

What does `bool()` return for each?

1. `bool(42)` — `True`
2. `bool(0)` — `False`
3. `bool("False")` — `True` (non-empty string!)
4. `bool("")` — `False`
5. `bool(" ")` — `True` (contains a space!)

Comparing Strings

Comparing Strings

```
>>> "apple" == "apple"
```

Comparing Strings

```
>>> "apple" == "apple"
True
>>> "apple" == "Apple"
False
>>> "hello" != "world"
True
```

Case matters!

`"apple"` and `"Apple"` are **different** strings

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
>>> "a" < "b"
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
>>> "a" < "b"
True
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
>>> "a" < "b"
True
>>> "A" < "a"
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
>>> "a" < "b"
True
>>> "A" < "a"
True
```

String Ordering

Strings are compared **character by character** (like a dictionary):

```
>>> "apple" < "banana"
True
>>> "a" < "b"
True
>>> "A" < "a"
True
```

Uppercase letters come **before** lowercase

`"A" < "Z" < "a" < "z"`

Common Mistakes

Pitfall #1: = vs ==

✗ `x = 5`

This **assigns** 5 to x

Pitfall #1: = vs ==

✗ `x = 5` This **assigns** 5 to x

✓ `x == 5` This **compares** x to 5

Pitfall #1: = vs ==

✗ `x = 5` This **assigns** 5 to x

✓ `x == 5` This **compares** x to 5

Pitfall #2: Case Sensitivity

✗ `x = true` `NameError: name 'true' is not defined`

✓ `x = True` Capital T!

Pitfall #3: Float Comparison

```
>>> 0.1 + 0.2 == 0.3
```

Pitfall #3: Float Comparison

```
>>> 0.1 + 0.2 == 0.3
```

`False` Wait, what?!

Pitfall #3: Float Comparison

```
>>> 0.1 + 0.2 == 0.3
False
```

Wait, what?!

```
>>> 0.1 + 0.2
0.30000000000000004
```

Pitfall #3: Float Comparison

```
>>> 0.1 + 0.2 == 0.3
False
```

Wait, what?!

```
>>> 0.1 + 0.2
0.30000000000000004
```

Pitfall #4: Range with wrong operator

"x is NOT between 1 and 10":

✗ `x < 1 and x > 10` (impossible! always False)

✓ `x < 1 or x > 10` (correct!)

Practice

Practice: Evaluate

Let `x = 4`, `y = 7`, `z = 2`

1. `x + z == y - 1`
2. `not (x > y) and z < x`
3. `x > z or y < z`
4. `x ** z > y and not (z == 2)`

Practice: Evaluate

Let `x = 4`, `y = 7`, `z = 2`

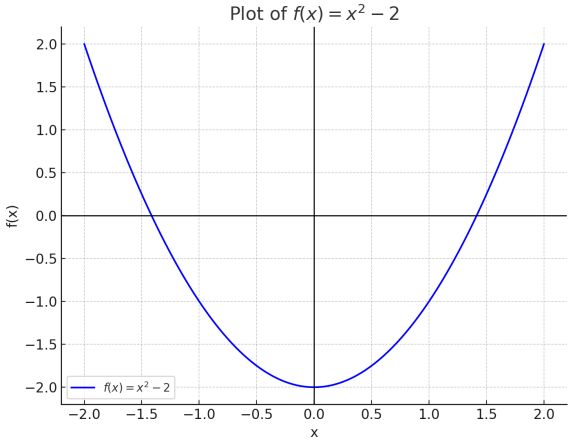
1. `x + z == y - 1` — `True` (6 == 6)
2. `not (x > y) and z < x` — `True` (True and True)
3. `x > z or y < z`
4. `x ** z > y and not (z == 2)`

Practice: Evaluate

Let `x = 4`, `y = 7`, `z = 2`

1. `x + z == y - 1` — `True` (6 == 6)
2. `not (x > y) and z < x` — `True` (True and True)
3. `x > z or y < z` — `True` (True or False)
4. `x ** z > y and not (z == 2)` — `False` (True and False)

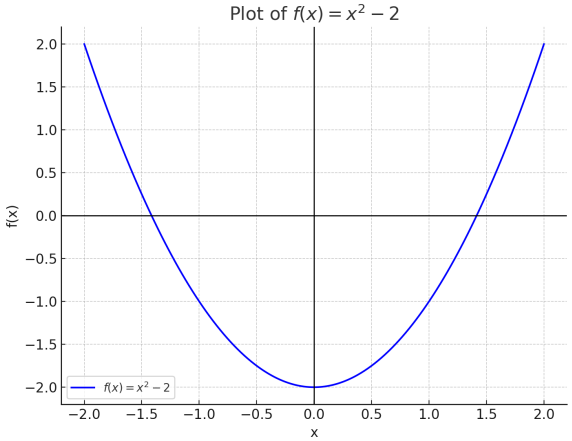
Reading a Graph



Looking at $f(x) = x^2 - 2$:

For which values of x is $f(x) > 0$?

Reading a Graph



Looking at $f(x) = x^2 - 2$:

For which values of x is $f(x) > 0$?

Answer: when $x < -\sqrt{2}$ or $x > \sqrt{2}$

In Python:

```
x**2 - 2 > 0
```

This is a boolean expression!

Fill in the Blank

What **operator** makes each expression `True`?

1. `5 ____ 3` is `True`
2. `(2 + 3) ____ 5` is `True`
3. `True ____ False` is `True`
4. `True ____ False` is `False`

Fill in the Blank

What **operator** makes each expression `True`?

1. `5 ____ 3` is `True` — `>`, `>=`, `!=`
2. `(2 + 3) ____ 5` is `True`
3. `True ____ False` is `True`
4. `True ____ False` is `False`

Fill in the Blank

What **operator** makes each expression `True`?

1. `5 ____ 3` is `True` — `>, >=, !=`
2. `(2 + 3) ____ 5` is `True` — `==, >=, <=`
3. `True ____ False` is `True`
4. `True ____ False` is `False`

Fill in the Blank

What **operator** makes each expression `True`?

1. `5 ____ 3` is `True` — `>, >=, !=`
2. `(2 + 3) ____ 5` is `True` — `==, >=, <=`
3. `True ____ False` is `True` — `or, !=`
4. `True ____ False` is `False`

Fill in the Blank

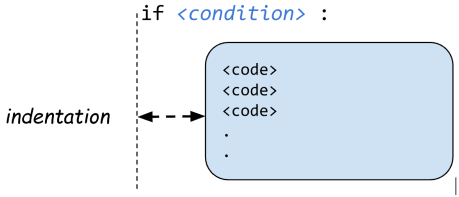
What **operator** makes each expression `True`?

1. `5 ____ 3` is `True` — `>, >=, !=`
2. `(2 + 3) ____ 5` is `True` — `==, >=, <=`
3. `True ____ False` is `True` — `or, !=`
4. `True ____ False` is `False` — `and, ==`

Why Do Booleans Matter?

Why Do Booleans Matter?

Because they power **decisions** in programs!



The <condition> is a **boolean expression**! Coming up next...

Summary

- `bool` type: only `True` or `False`
- **Comparison operators:** `==` `!=` `<` `>` `<=` `>=`
- **Logical operators:** `not`, `and`, `or`
- **Precedence:** arithmetic → comparison → `not` → `and` → `or`
- **Truthy/Falsy:** `0`, `""`, `None` are `False`; most other values are `True`
- **Strings** can be compared with ==, <, etc.
- When in doubt, use **parentheses**!

Questions?