Deck 004
Python – Operators

Dr. Hazel “twitch.tv/hazeldotzone” Campbell

Things we've seen

`int`	`*`	`int`	multiplication
`str`	`*`	`int`	repetition
`int`	`+`	`int`	addition
`str`	`+`	`str`	concatenation
`function`	`()`		function call
`type`	`()`		make something that type
	`()`		delimited expression (parentheses/brackets)

Things we've seen

`int`	`*`	`int`	multiplication
`str`	`*`	`int`	repetition
`int`	`+`	`int`	addition
`str`	`+`	`str`	concatenation
`callable`	`()`		function call / type call
	`()`		delimited expression (parentheses/brackets)

The Reference

https://docs.python.org/3/reference/expressions.html

6.17. Operator precedence

Operator	Description
`(expressions...)`, `[expressions...]`, `{key: value...}`, `{expressions...}`	Binding or parenthesized expression, list display, dictionary display, set display
`x[index]`, `x[index:index]` `x(arguments...)`, `x.attribute`	Subscription (including slicing), call, attribute reference
`await x`	Await expression
`**`	Exponentiation [5]
`+x`, `-x`, `~x`	Positive, negative, bitwise NOT
`*`, `@`, `/`, `//`, `%`	Multiplication, matrix multiplication, division, floor division, remainder [6]
`+`, `-`	Addition and subtraction

P

E

MD

AS

Things we've seen

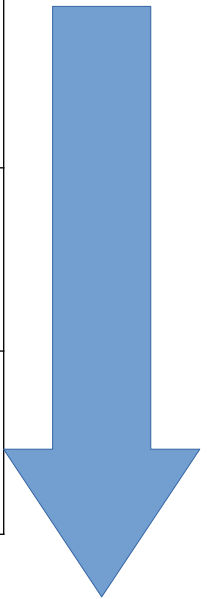
`int`	`*`	`int`	multiplication
`str`	`*`	`int`	repetition
`int`	`+`	`int`	addition
`str`	`+`	`str`	concatenation
`callable`	`()`		function call / type call
	`()`		delimited expression (parenthesized)

Let's focus on just numbers

	`()`		delimited expression (parenthesized)
`int`	`*`	`int`	multiplication
`int`	`+`	`int`	addition

Let's focus on just numbers

	`()`		delimited expression (parenthesized)
`int`	`*`	`int`	multiplication
`int`	`+`	`int`	addition

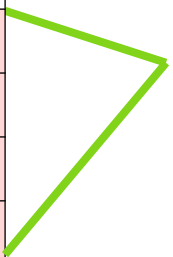


evaluate first

evaluate last

More operators

in type	characters	in type	result	operation	Evaluation order
	( )			delimited expression	1st
int	**	int	int	exponentiation	2nd
	+	int	int	positive	3rd
	-	int	int	negative / negation	3rd
int	*	int	int	multiplication	4th
int	/	int	float	floating point division	4th
int	//	int	int	floor division	4th
int	%	int	int	remainder	4th
int	+	int	int	addition	5th
int	-	int	int	subtraction	5th



Same precedence evaluate left-to-right

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

P
E

MD

AS

Same precedence
always evaluate
left-to-right

Same precedence
always evaluate
left-to-right

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"Parentheses"
"Brackets"

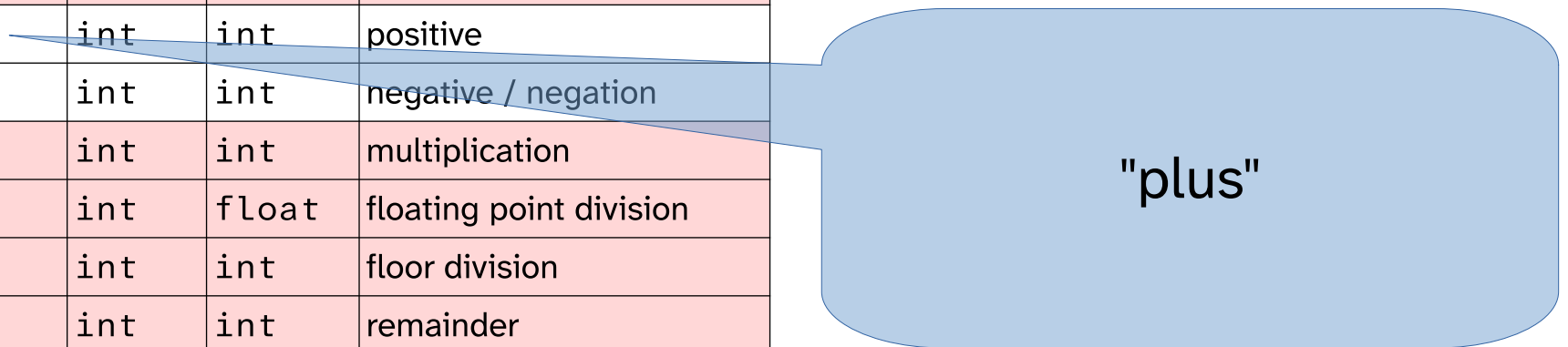
More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"star star"
"asterisk asterisk"

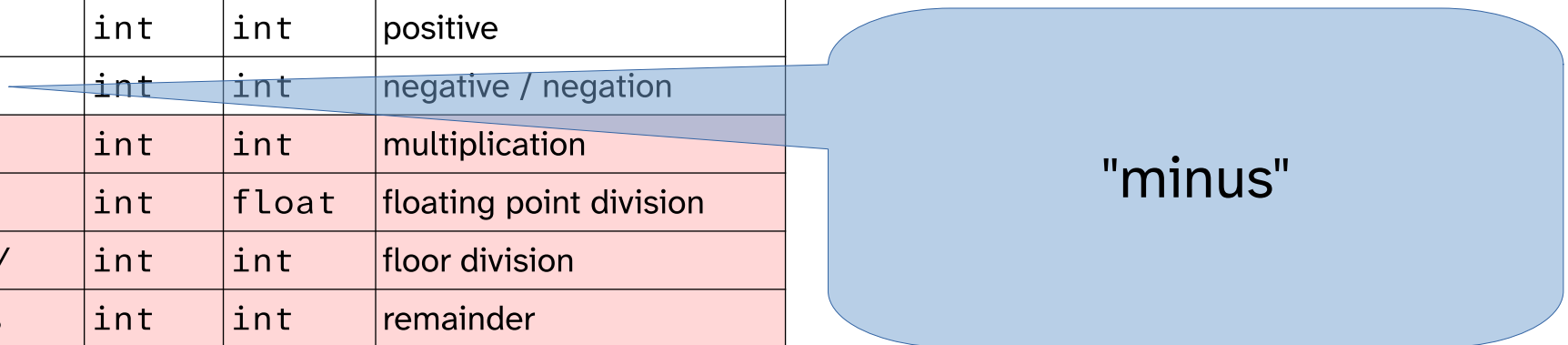
More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction



More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction



More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"star"
"asterisk"

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"slash"
"forward slash"

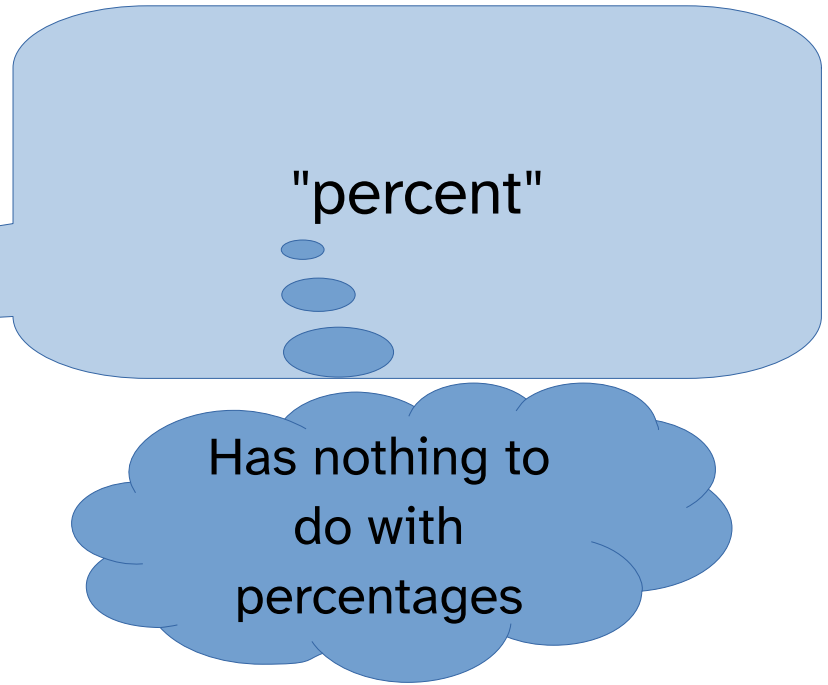
More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"slash slash"

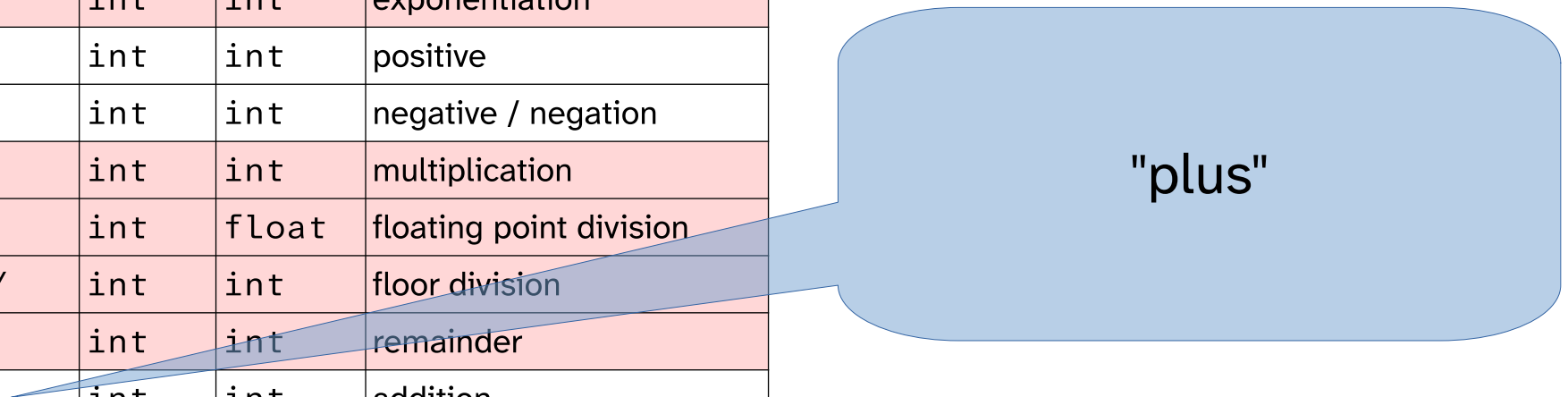
More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction



More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction



More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

"minus"

Same Precedence

8 - 2 + 3

Both operators are the same precedence,
so we evaluate them left to right

Same Precedence

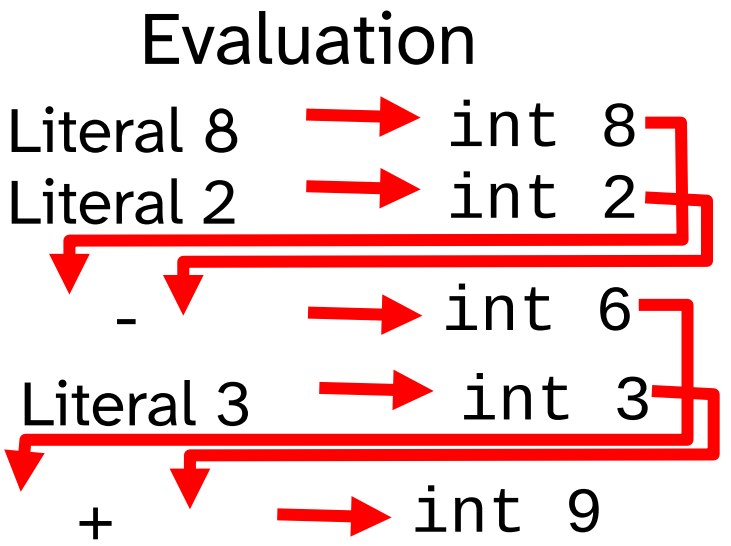
first second

8 - 2 + 3

Same Precedence

first second

```
8 - 2 + 3
```

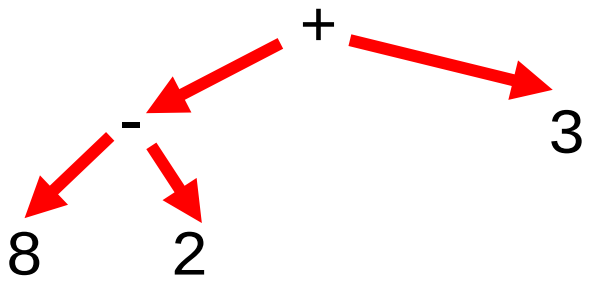


Same Precedence

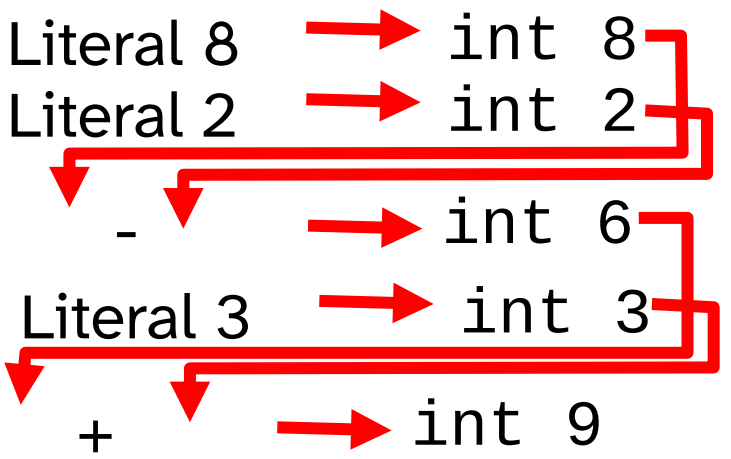
first second

```
8 - 2 + 3
```

Syntax Diagram



Evaluation



Same Precedence

```
8 / 2 * 3
```

Both operators are the same precedence,
so we evaluate them left to right

Same Precedence

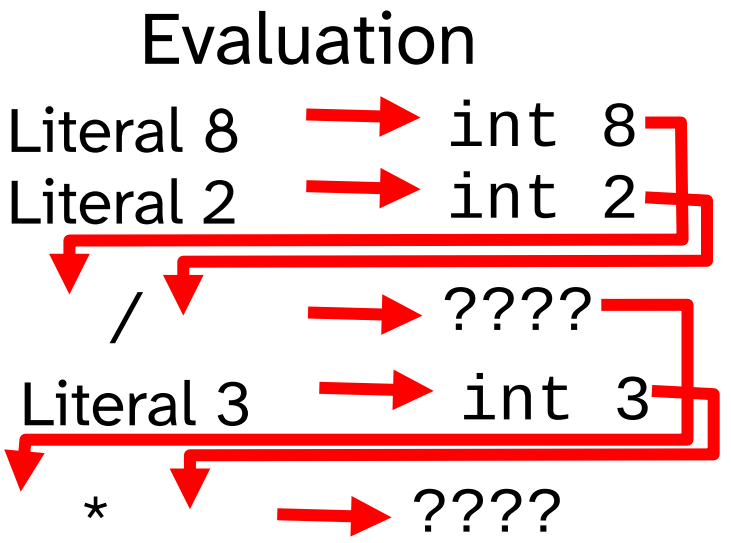
first second

```
8 / 2 * 3
```

Same Precedence

first second

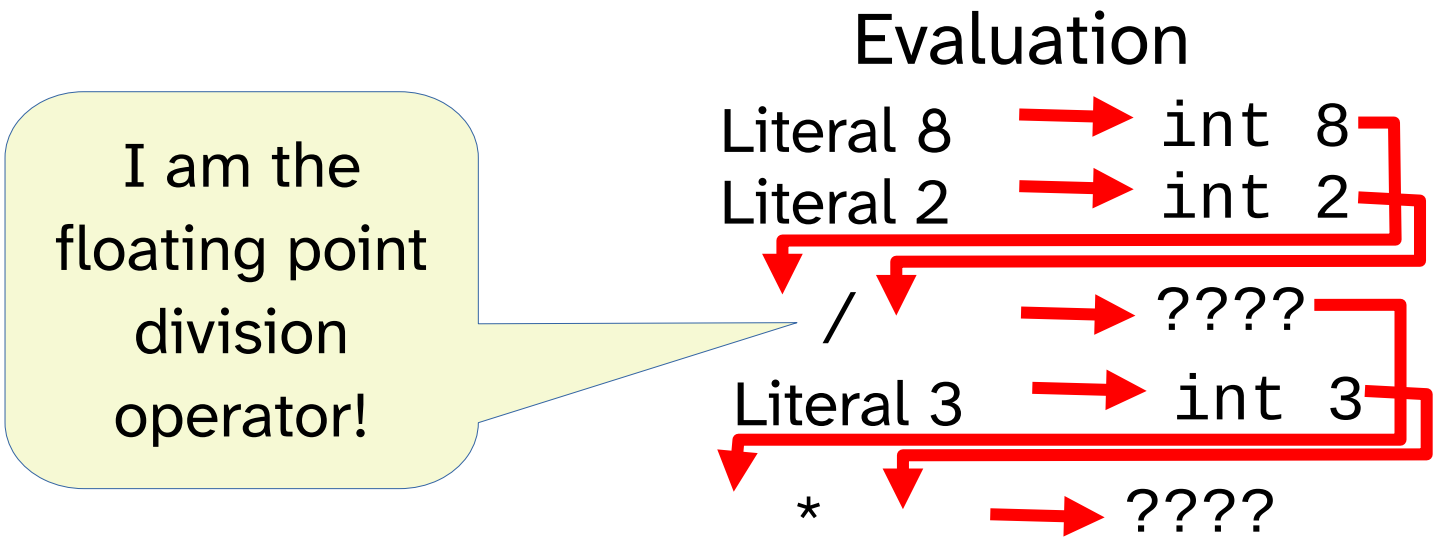
```
8 / 2 * 3
```



Same Precedence

first second

```
8 / 2 * 3
```



float

```
>>> 8/2
4.0
>>>
```

```
>>> type(8/2)
<class 'float'>
>>>
```

- `float` is a totally different type from `int`. It's a double-precision floating point number:
  a number that can have non-integer value
  - Decimals
    - Limit 15 significant figures!
  - Fractions????? Kinda

/!\ float /!\

```
>>> 1/100
0.01
>>> 1/(1/100)
100.0
```

ok

- Every operation on a float is rounded in... some way...

```
>>> 1/103
0.009708737864077669
>>> 1/(1/103)
103.00000000000001
```

Not the same!

- Every operation on a float is rounded in... some way...

float

```
>>> 8/2
4.0
>>>
```

```
>>> type(8/2)
<class 'float'>
>>>
```

- `float` is a totally different type from `int`. It's a double-precision floating point number:
  a number that can have non-integer value
  - Decimals
    - Limit 15 significant figures!
  - Fractions????? Kinda

float

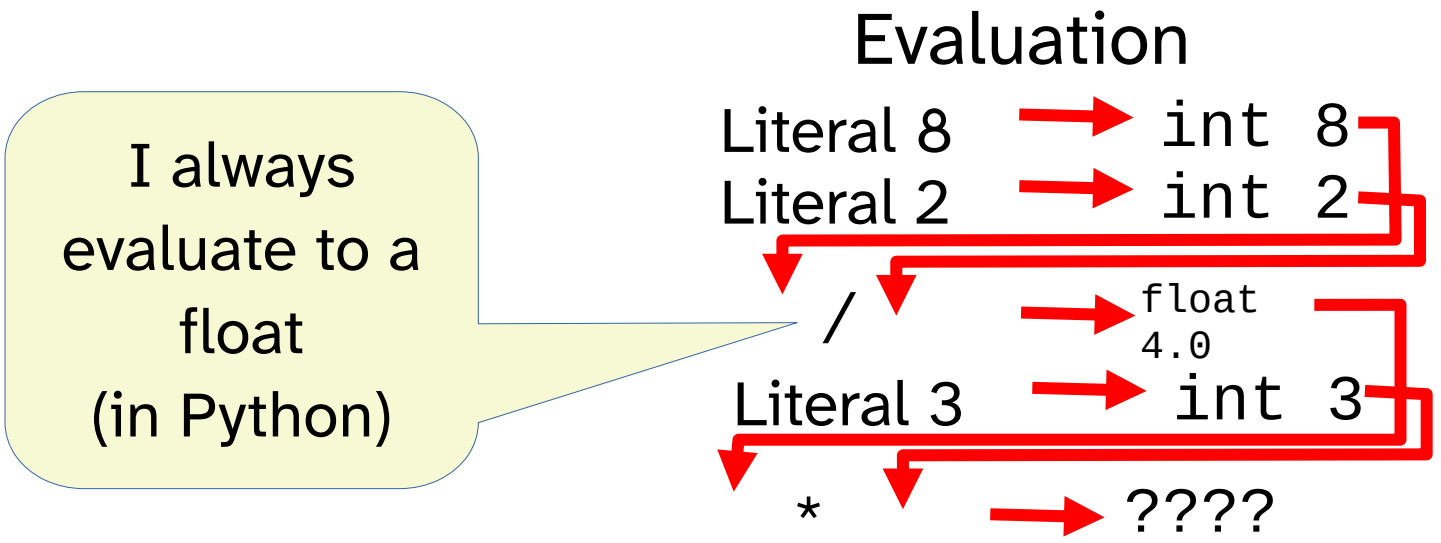
```
>>> float(1)
1.0
>>> float("1")
1.0
>>> float("1.5")
1.5
>>> float("1e3")
1000.0
```

```
>>> type(1)
<class 'int'>
>>> type(1.0)
<class 'float'>
```

Same Precedence

first second

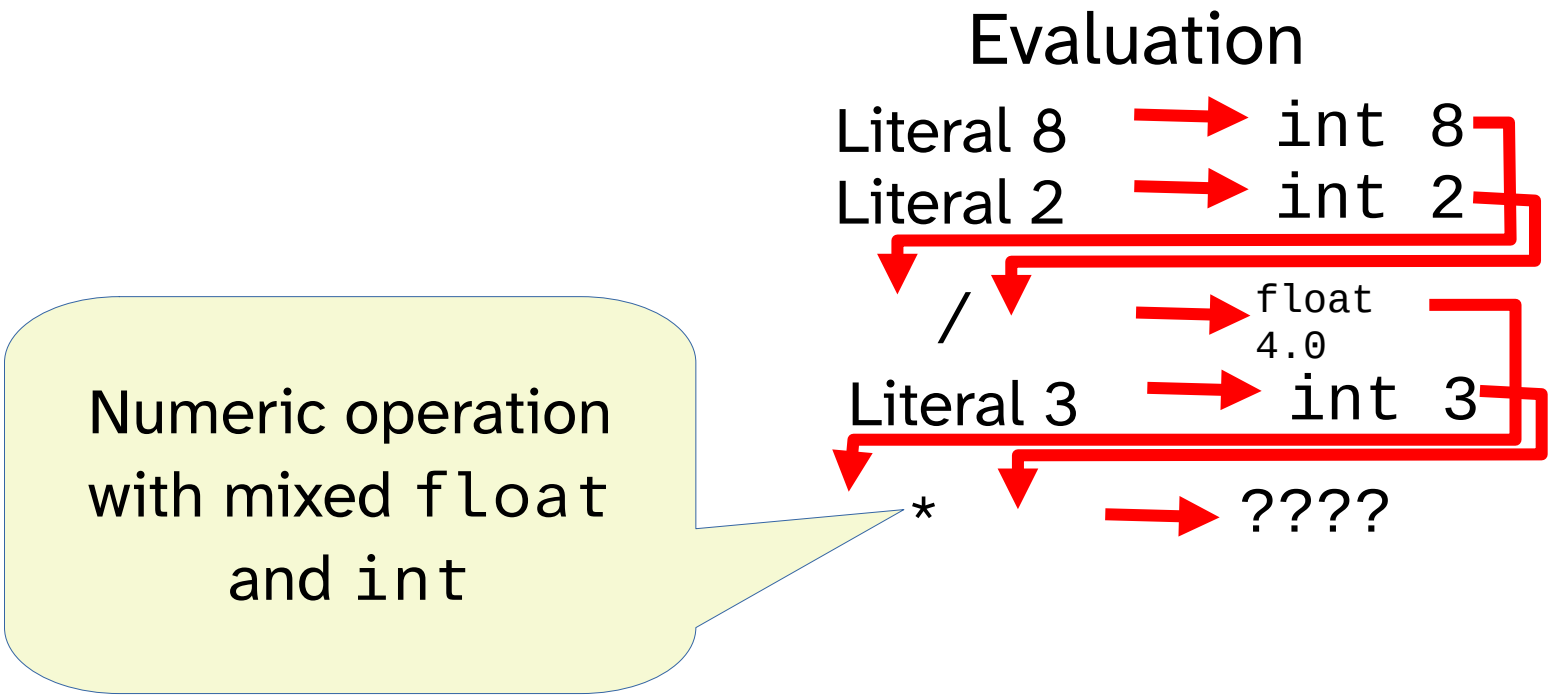
```
8 / 2 * 3
```



Same Precedence

first second

```
8 / 2 * 3
```



Same Precedence

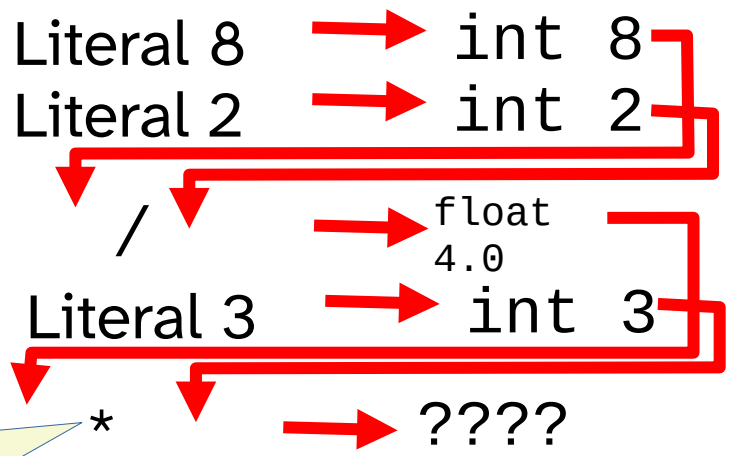
first second

```
8 / 2 * 3
```

```
>>> 4.0*3
12.0
>>> 8/2*3
12.0
```

Numeric operation with mixed `float` and `int` becomes a `float`

Evaluation



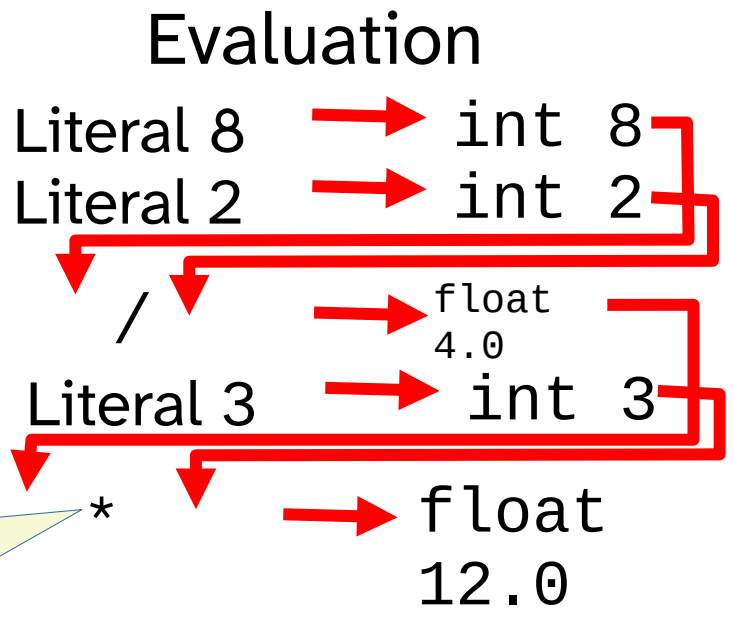
Same Precedence

first second

```
8 / 2 * 3
```

```
>>> 4.0*3
12.0
>>> 8/2*3
12.0
```

Numeric operation with mixed `float` and `int` becomes a `float`



`float` literal

- A float literal starts with a number and contains a `.` or `e`

```
>>> type(12.0)
<class 'float'>
>>> type(12e3)
<class 'float'>
>>> type(6.66e9)
<class 'float'>
```

`float` literal

- The e stands for $\times 10^{\text{whatever}}$

```
>>> 12e3
12000.0
>>> 6.66e9
6660000000.0
```

e3 → thousand
e6 → million
e9 → billion
e-3 → thousandth

```
>>> 1e3
1000.0
>>> 1e-3
0.001
```

// the floor division operator

Doesn't exist in most other programming languages

```
>>> 7/3                    >>> type(7/3)
2.3333333333333335       <class 'float'>
>>> 7//3                   >>> type(7//3)
2                        <class 'int'>
```

In Python the `//` `int` division operator always rounds down

// the floor division operator

Doesn't exist in most other programming languages

```
>>> 7/3
2.3333333333333335
>>> 7//3
2
>>> -7/3
-2.3333333333333335
>>> -7//3
-3
```

In Python the `//` `int` division operator always rounds down

% the remainder operator

Does division and evaluates to the remainder

```
>>> 12345 % 100
45
>>> 12345.67 % 100.0
45.670000000000007
>>> 4 % 3
1
```

% the remainder operator

Does division and evaluates to the remainder

```
>>> 12345 % 100
45
>>> -12345 % -100
-45
>>> -12345 % 100
55
>>> 12345 % -100
-55
```

For %, the denominator determines the sign of the result (the sign of the remainder)

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
float	**	float	float	exponentiation
	+	float	float	positive
	-	float	float	negative / negation
float	*	float	float	multiplication
float	/	float	float	floating point division
float	//	float	float	floor division
float	%	float	float	remainder
float	+	float	float	addition
float	-	float	float	subtraction

More numerical operators

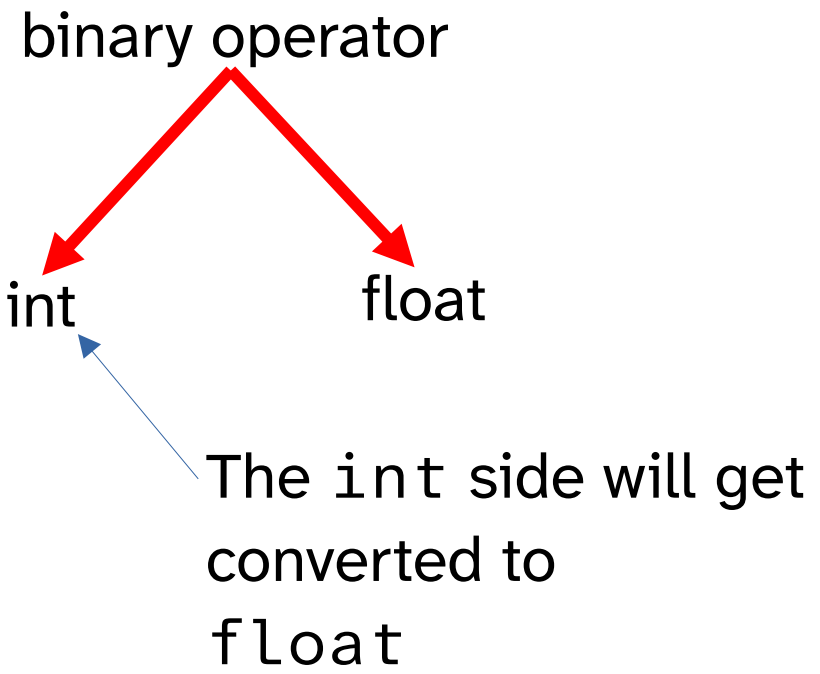
in type	characters	in type	result	operation
	( )			delimited expression
`float`	`**`	`float`	`float`	exponentiation
	`+`	`float`	`float`	positive
	`-`	`float`	`float`	negative / negation
`float`	`*`	`float`	`float`	multiplication
`float`	`/`	`float`	**`float`**	floating point division
`float`	`//`	`float`	`float`	floor division
`float`	`%`	`float`	`float`	remainder
`float`	`+`	`float`	`float`	addition
`float`	`-`	`float`	`float`	subtraction

Single slash
floating point division
is the only operation where
the type of the result
doesn't change between
`int` and `float`

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	float	float	exponentiation
	+	float	float	positive
	-	float	float	negative / negation
int	*	float	float	multiplication
int	/	float	float	floating point division
int	//	float	float	floor division
int	%	float	float	remainder
int	+	float	float	addition
int	-	float	float	subtraction

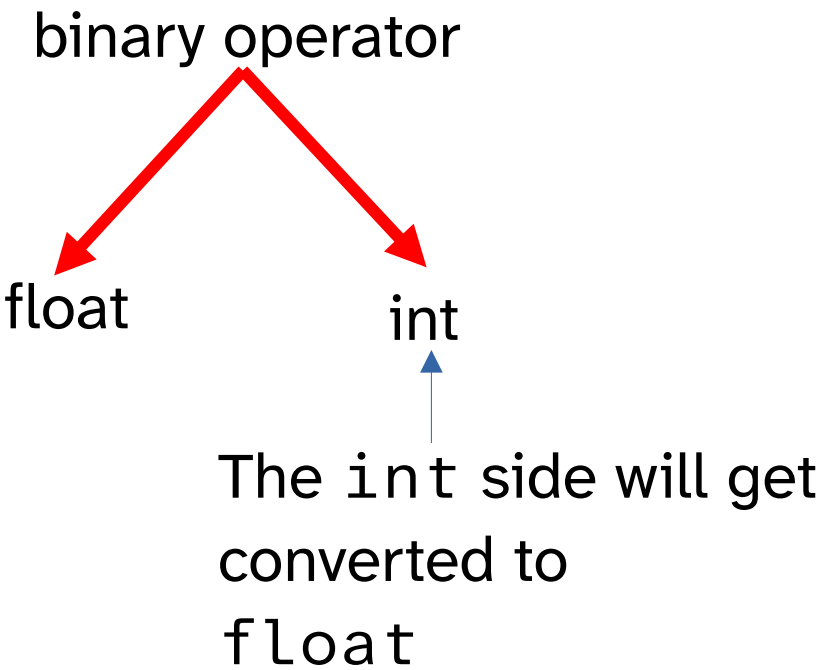
Well except float division



More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
float	**	int	float	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
float	*	int	float	multiplication
float	/	int	float	floating point division
float	//	int	float	floor division
float	%	int	float	remainder
float	+	int	float	addition
float	-	int	float	subtraction

Well except float division



Unary Operators

- Unary → Only has one operand (on the right)

```
>>> +7
7
>>> +(7)
7
>>> +(4+3)
7
```

```
>>> -7
-7
>>> -(7)
-7
>>> -(4+3)
-7
```

Unary Operators

- Unary → Only has one operand (on the right)

```
>>> +7
7
>>> +(7)
7
>>> +(4+3)
7
```

```
>>> -7
-7
>>> -(7)
-7
>>> -(4+3)
-7
```

Unary Operators

- Higher precedence than * / // % and + -

```
>>> -(4+3)
-7
>>> -(4)+(3)
-1
```

More numerical operators

in type	characters	in type	result	operation
	( )			delimited expression
`int`	`**`	`int`	`int`	exponentiation
	`+`	`int`	`int`	positive
	`-`	`int`	`int`	negative / negation
`int`	`*`	`int`	`int`	multiplication
`int`	`/`	`int`	`float`	floating point division
`int`	`//`	`int`	`int`	floor division
`int`	`%`	`int`	`int`	remainder
`int`	`+`	`int`	`int`	addition
`int`	`-`	`int`	`int`	subtraction

P
E

U for unary?

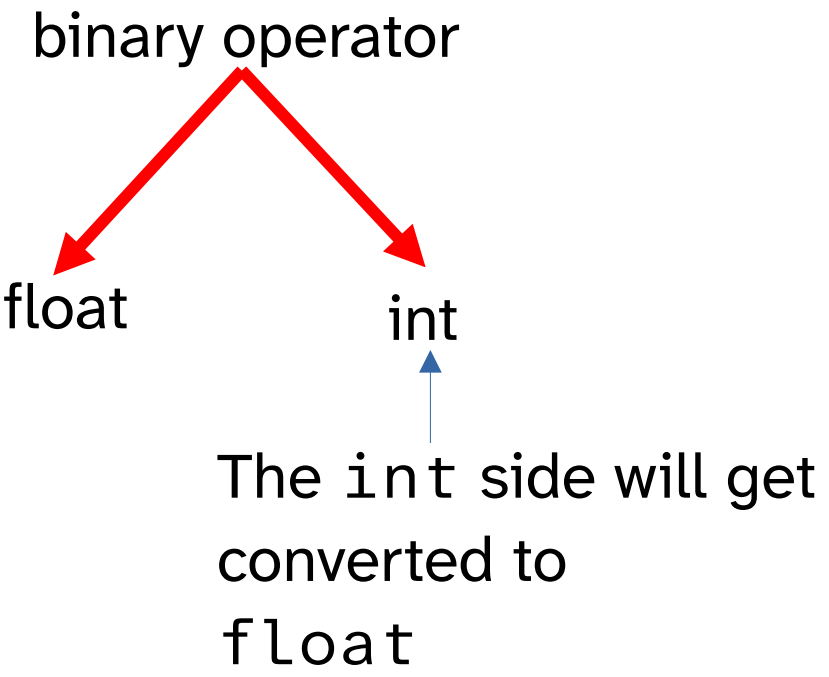
MD

AS

PEUMDAS?

More numerical operators

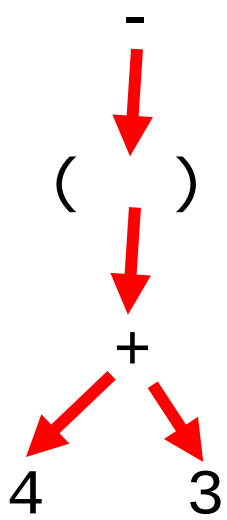
```
>>> 1+2.0
3.0
>>> 1.0+2
3.0
```



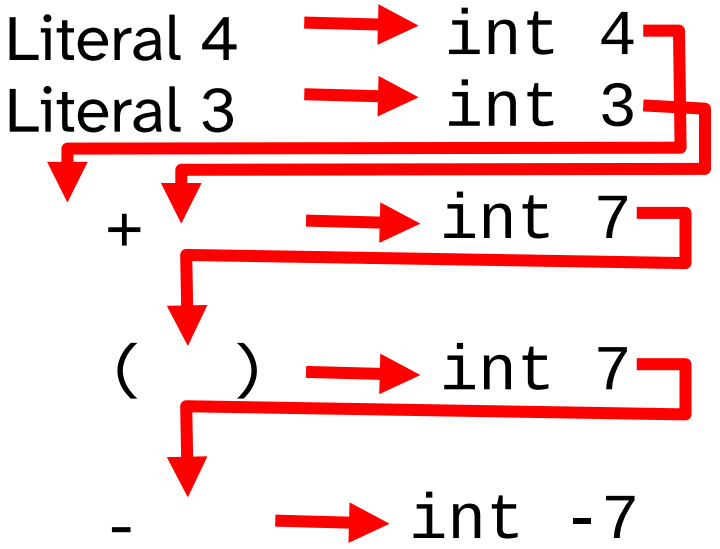
Unary Operators

```
>>> -(4+3)
-7
```

Syntax Diagram



Evaluation



Unary Operators

```
>>> +(4+3)
7
```

Unary plus usually does nothing

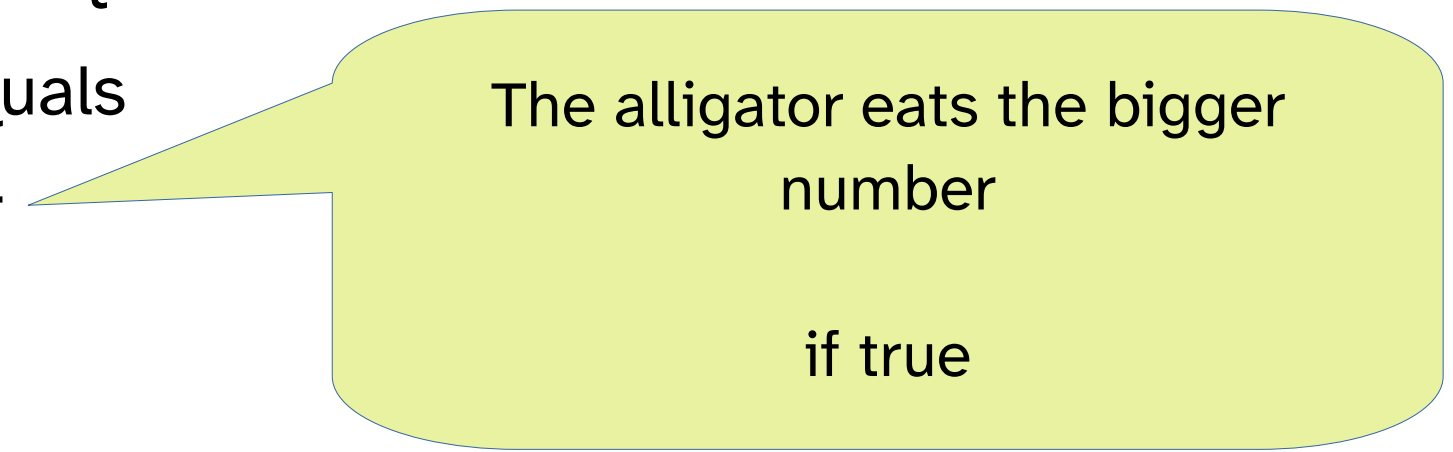
∴ we shouldn't use it, because it's confusing and unusual

Remember Software Engineering!

Avoid non-idiomatic or confusing code!

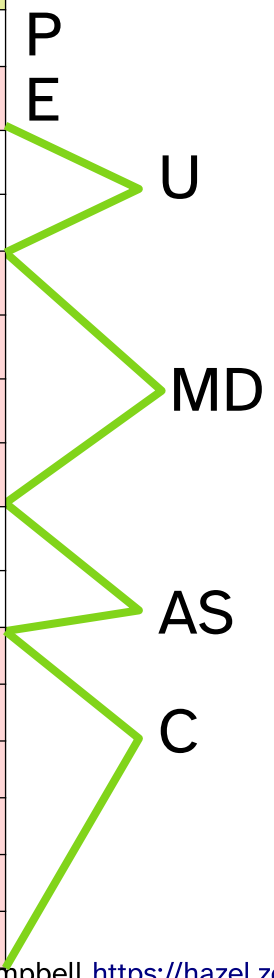
Comparison Operators

- == equals
- != unequals
- >= greater equals
- <= less equals
- > greater
- < less



Comparison operators

in type	characters	in type	result	operation
	( )			delimited expression
int	**	int	int	exponentiation
	+	int	int	positive
	-	int	int	negative / negation
int	*	int	int	multiplication
int	/	int	float	floating point division
int	//	int	int	floor division
int	%	int	int	remainder
int	+	int	int	addition
int	-	int	int	subtraction
int	<	int	bool	less than
int	<=	int	bool	less than equal
int	>	int	bool	greater than
int	>=	int	bool	greater than equal
int	==	int	bool	equal
int	!=	int	bool	unequal



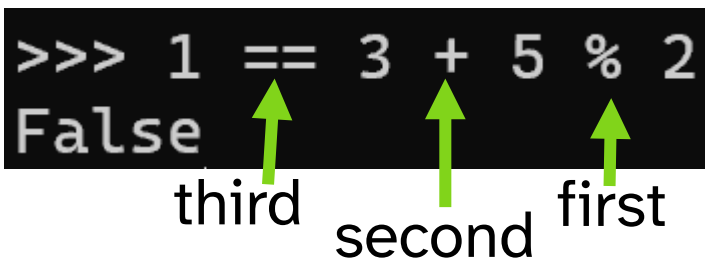
PEUMDASC?

Comparison operators are lower precedence (evaluated after) addition and subtraction

Comparison operators

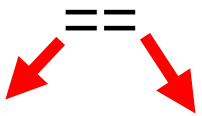
```
>>> 1 == 3 + 5 % 2
False
```

Comparison operators



Comparison operators

Syntax Diagram



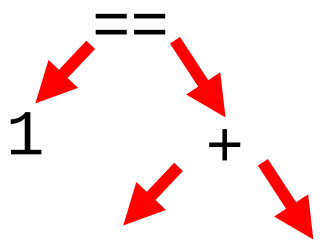
Lowest precedence
(last)
goes at the top

```
>>> 1 == 3 + 5 % 2
False
```

third second first

Comparison operators

Syntax Diagram



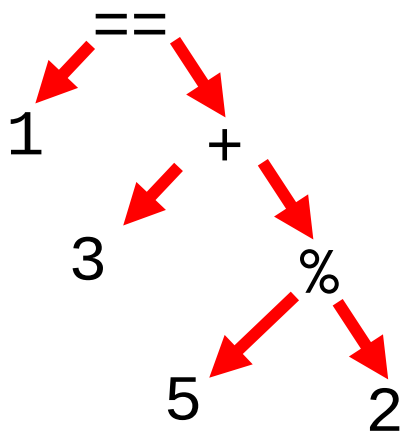
Lowest precedence
(last)
goes at the top

```
>>> 1 == 3 + 5 % 2
False
```

third second first

Comparison operators

Syntax Diagram



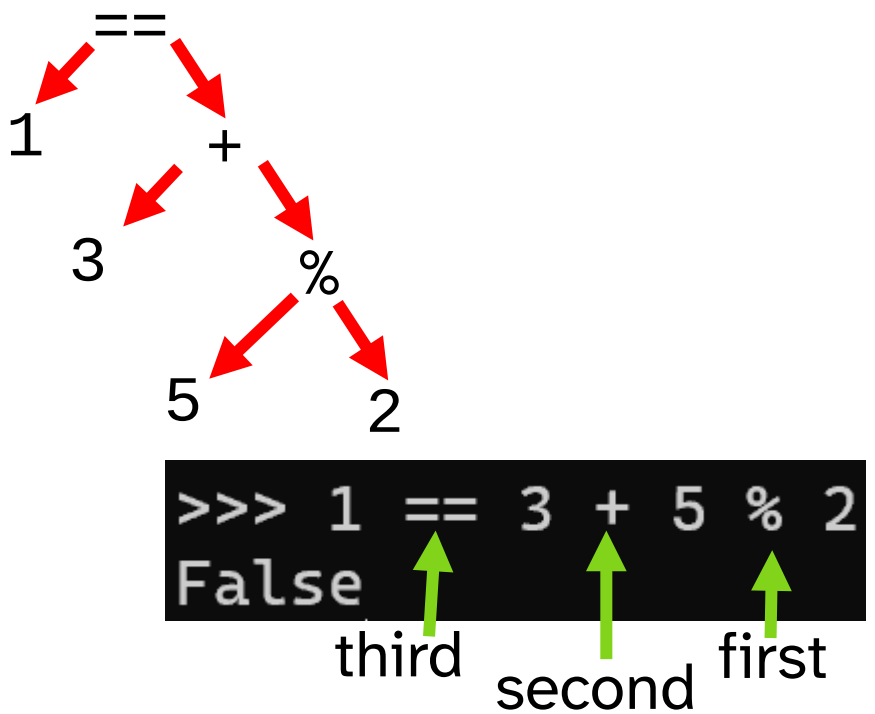
Lowest precedence
(last)
goes at the top

```
>>> 1 == 3 + 5 % 2
False
```

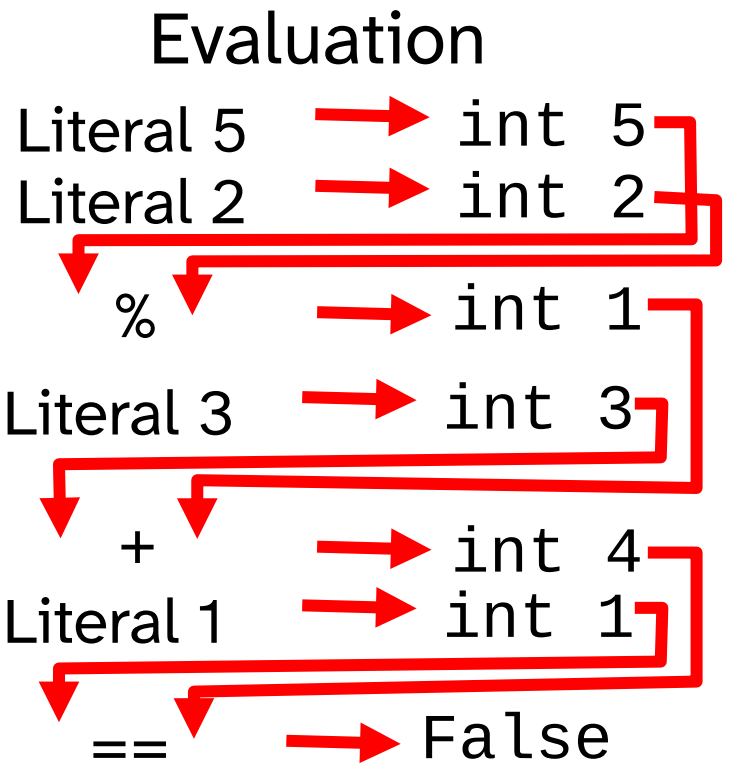
third second first

Comparison operators

Syntax Diagram



Evaluation



Exponentiation operator

```
>>> 2**10
1024
>>> 2**0.5
1.4142135623730951
>>> 1.4142135623730951**10
32.000000000000002
>>> 1.4142135623730951**20
1024.0000000000014
```

2^{10}

$\sqrt{2}$

$\left(\sqrt{2}\right)^{10}$

$\left(\sqrt{2}\right)^{20}$

Exponentiation operator

```
>>> 2**500
3273390607896141870013189696827599152216642046043064789483291368096133796404674554883270092325904157150886684127560071009217256545885393053328527589376
>>> 2.0**500
3.273390607896142e+150
```

In Python, there is no limit to the number of digits in an int

(but there is a limit to to the number of digits in a float...)

Put Some Things Together

```
>>> -(123+500)==-123-500
True
```

The – unary operator is the same as multiplying by -1, so it distributes as expected.

Put Some Things Together

123 is 12 × 10 + 3