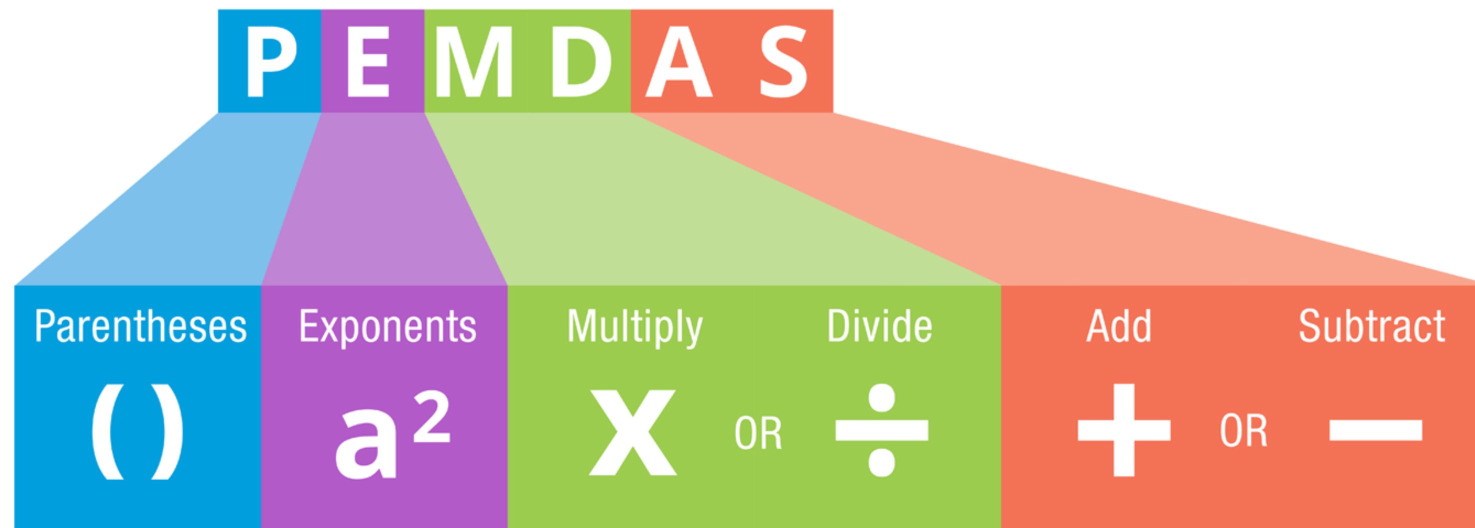


CSCI 141

Scott Wehrwein

Operators and Operands
Order of Operations

Goals

- Know the definition and usage of operators and operands

- Know the behavior and purpose of each of the following operators:
  `=, +, -, *, **, /, //, %`

- Know how to apply operator precedence rules to determine the order in which pieces of an expression are evaluated.

Operators

- Operators are special symbols that represent computations we can perform.
- Operands are the values that an operator performs its computations on.
- We've seen one already: the assignment operator.

Its first (left) operand

Its second (right) operand

```
my_age = 32
```

The assignment operator.

Operators

Some more Python operators:

```
=
+
-
*
/
**
//
%
```

Some of these probably look familiar…

Operators

Some more Python operators:

= Assignment operator: stores a value in a variable

+ Addition

− Subtraction

* Multiplication

These ones do exactly what you think.

/ Division

**

//

%

Operators

Some more Python operators:

- `=` Assignment operator: stores a value in a variable
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**`
- `//`
- `%`

This one too, with one quirk:
In Python, division **always** returns a `float`.

```
3.0 / 2 => 1.5
7 / 2   => 3.5
4 / 2   => ??
```

Operators

Some more Python operators:

- `=` Assignment operator: stores a value in a variable
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**`
- `//`
- `%`

This one too, with one quirk:
In Python, division **always** returns a `float`.

```
3.0 / 2 => 1.5
7 / 2   => 3.5
4 / 2   => 2.0
```

Operators

Some more Python operators:

- `=` Assignment operator: stores a value in a variable
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**` Exponentiation
- `//`
- `%`

The exponentiation operator raises the left operand to the power of the right operand.

Math: $2^4 = 2 * 2 * 2 * 2 = 16$

Python: `2**4 => 16`

Base Exponent

Operators

Some more Python operators:

- `=` Assignment operator: stores a value in a variable
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**` Exponentiation
- `//` Integer division
- `%` Modulus (remainder)

Integer division does division and evaluates to the integer **quotient**

Math: 7 / 2 is 3 with remainder 1

Python: `7 // 2 => 3`

Operators

Some more Python operators:

- `=` Assignment operator: stores a value in a variable
- `+` Addition
- `-` Subtraction
- `*` Multiplication
- `/` Division
- `**` Exponentiation
- `//` Integer division
- `%` Modulus (remainder)

The modulus operator does division and evaluates to the integer **remainder**

Math: 7 / 2 is 3 with remainder 1

Python: `7 % 2 => 1`

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules?

What if we took the parentheses out?

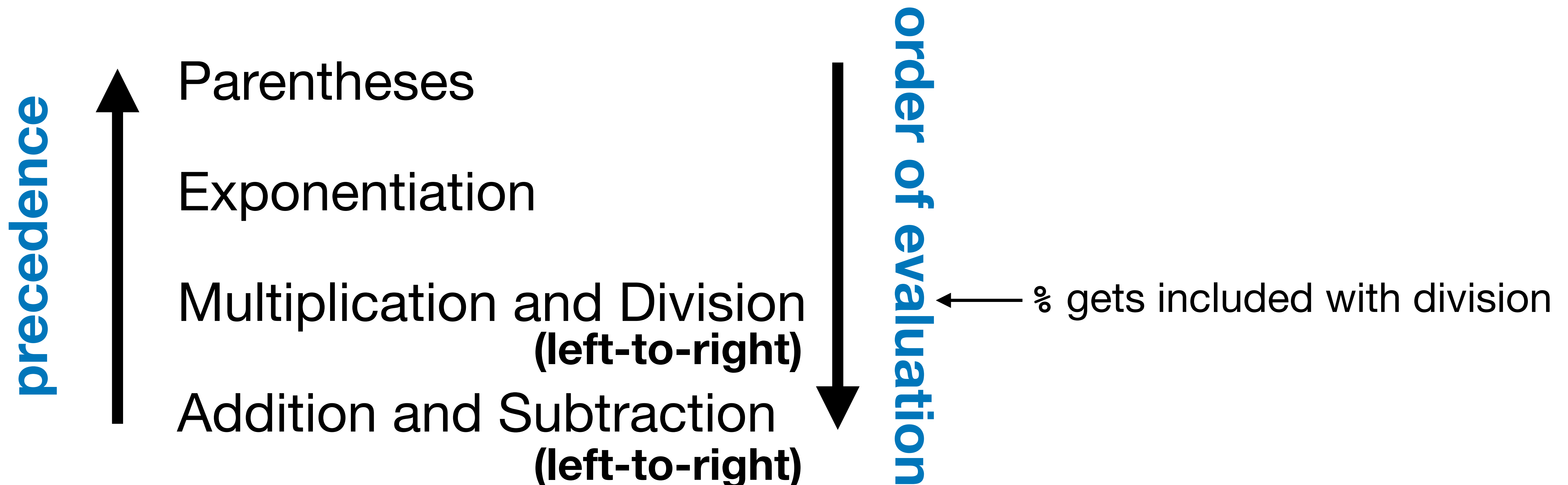
```
result = 5 % (3 ** (6 // 4))

result = 5 % 3 ** 6 // 4
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

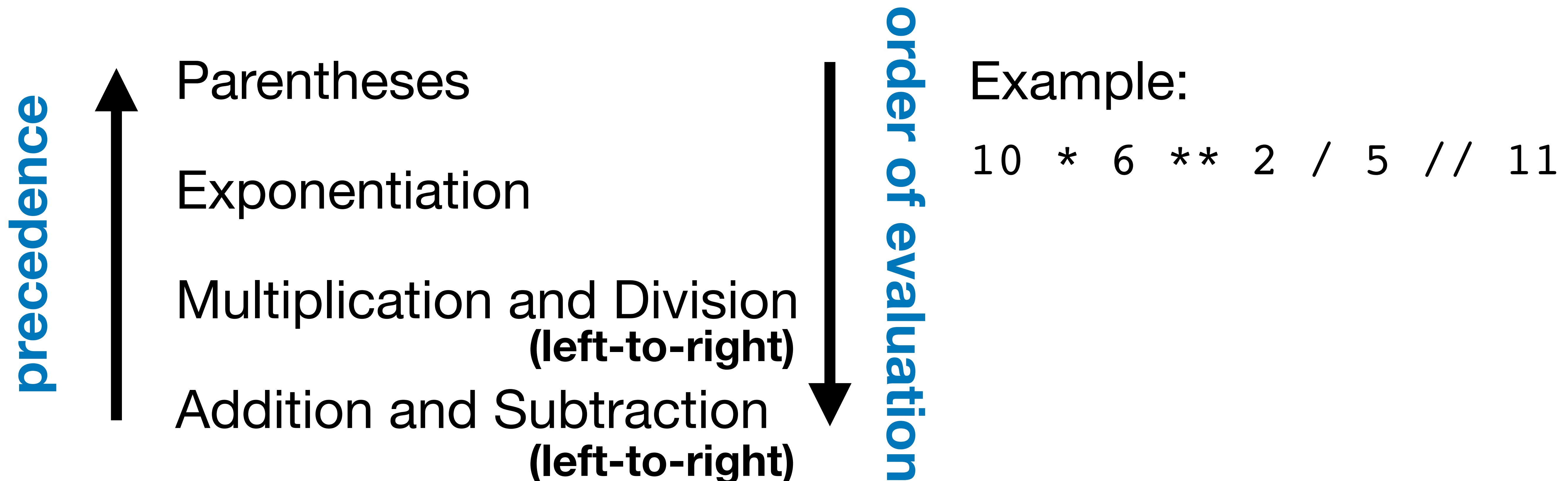
Remember PEMDAS? BIDMAS? BODMAS?



Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

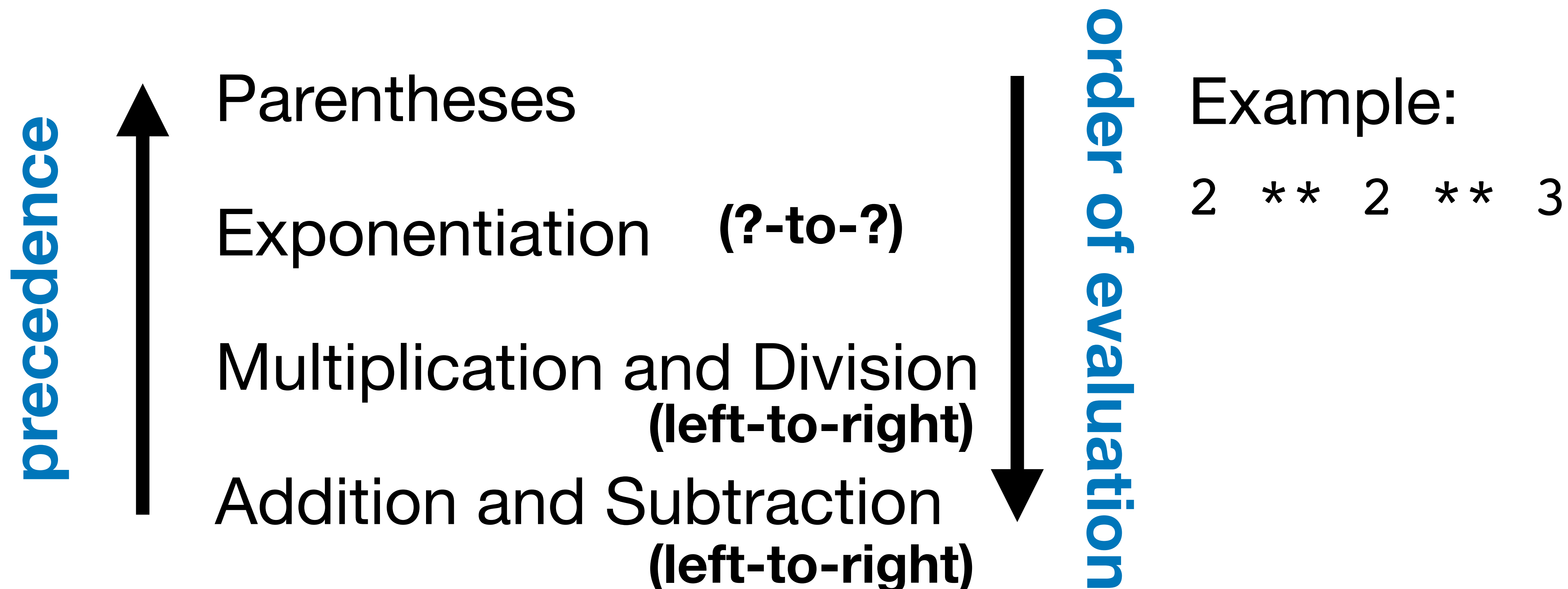
Remember PEMDAS? BIDMAS? BODMAS?



Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?



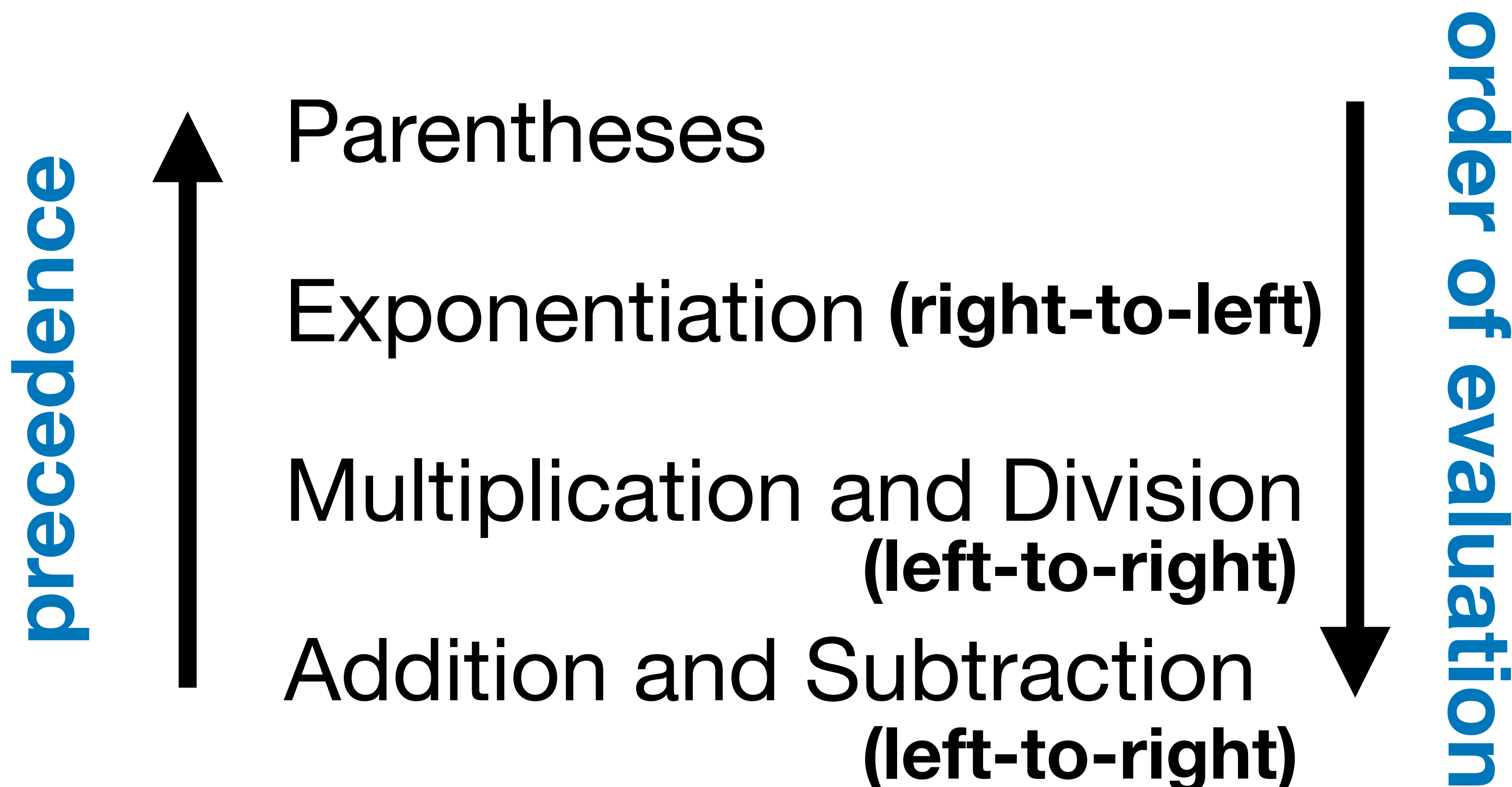
Example:

```
2 ** 2 ** 3
```

Order of Operations

We know parenthesized expressions get evaluated from inside to out. Are there any other rules? Yes: operator precedence.

Remember PEMDAS? BIDMAS? BODMAS?



Example:

```
2 ** 2 ** 3

(2 ** 2) ** 3
=> 4³ => 64

2 ** (2 ** 3)
=> 2⁸ => 256
```

Types of operands

- Operators only work if their operands have the correct types. `float * str => error`
- Some operators can work on more than one type or combination of types:

Not too surprising:

```
int + int => int
int + float => float
float + int => float
float + float => float
```

Maybe a little surprising:

```
str + str => str
str * int => str
```

Demo

Demo

- operator behaviors:

```
4 + 5 => 9
4.0 + 5 => 9.0
4.0 + 5.0 => 9.0
“a” + “b” => “ab”
“a” + 1 => error
“a” + “b” => “ab”
“a” * 16 => “aaaaaaaaaaaaaaaa”
```