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
/ 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.

\[
\begin{align*}
3.0 / 2 & \Rightarrow 1.5 \\
7 / 2 & \Rightarrow 3.5 \\
4 / 2 & \Rightarrow ??
\end{align*}
\]
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: $2^4 = 2 \times 2 \times 2 \times 2 = 16$

Python: $2**4 \Rightarrow 16$

Base \hspace{1cm} Exponent
Operators

Some more Python operators:

=  Assignment operator: stores a value in a variable
+
-  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?

- Parentheses
- Exponentiation
- Multiplication and Division (left-to-right)
- Addition and Subtraction (left-to-right)

% gets included with division
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:

10 * 6 ** 2 / 5 // 1

1

precedence

Parentheses
Exponentiation
Multiplication and Division (left-to-right)
Addition and Subtraction (left-to-right)

order of evaluation
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: $2^2^3$

Precedence

Parentheses
Exponentiation (left-to-right)
Multiplication and Division (left-to-right)
Addition and Subtraction (left-to-right)

Order of Evaluation

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:**

\[
2 \ ** \ 2 \ ** \ 3 \\
(2 \ ** \ 2) \ ** \ 3 \\
2 \ ** \ (2 \ ** \ 3)
\]

<table>
<thead>
<tr>
<th>First</th>
<th>Second</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 \ ** \ 2 \ ** \ 3</td>
<td>=&gt; (4^3) =&gt; 64</td>
<td></td>
</tr>
<tr>
<td>(2 \ ** \ 2) \ ** \ 3</td>
<td>=&gt; (4^3) =&gt; 64</td>
<td></td>
</tr>
<tr>
<td>2 \ ** \ (2 \ ** \ 3)</td>
<td>=&gt; (2^8) =&gt; 256</td>
<td></td>
</tr>
</tbody>
</table>
Types of operands

• Operators only work if their operands have the correct types. $\text{float} \ast \text{str} \Rightarrow \text{error}$

• Some operators can work on more than one type or combination of types:

Not too surprising:

\begin{align*}
\text{int} + \text{int} & \Rightarrow \text{int} \\
\text{int} + \text{float} & \Rightarrow \text{float} \\
\text{float} + \text{int} & \Rightarrow \text{float} \\
\text{float} + \text{float} & \Rightarrow \text{float}
\end{align*}

Maybe a little surprising:

\begin{align*}
\text{str} + \text{str} & \Rightarrow \text{str} \\
\text{str} \ast \text{int} & \Rightarrow \text{str}
\end{align*}
Demo
Demo

• operator behaviors:

  $4 + 5 \Rightarrow 9$
  $4.0 + 5 \Rightarrow 9.0$
  $4.0 + 5.0 \Rightarrow 9.0$
  “a” + “b” \Rightarrow “ab”
  “a” + 1 \Rightarrow error
  “a” + “b” \Rightarrow “ab”
  “a” * 16 \Rightarrow “aaaaaaaaaaaaaaaaaaaaa”