1 Arithmetic Expressions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Operation</th>
<th>Expression</th>
<th>English description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>addition</td>
<td>11 + 56</td>
<td>11 plus 56</td>
<td>67</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>12 - 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>4 * 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>2 ** 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/</td>
<td>division</td>
<td>8 / 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>remainder</td>
<td>8 % 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q. With 8 % 3 we get the remainder from “long division”, but 8 / 3 doesn’t give us the long-division quotient. How do we get the quotient?

A.

Expressions with multiple operators:

Evaluate the following expressions:

- 10 + 2 * 3 + 4
- 10 * 2 + 3 * 4

List the basic operators in order of precedence:

Q. What happens when we try to evaluate an expression that can’t work (e.g., 4 / 0)?

A.

2 Python Types

Let’s use Python’s function type to find out the types of the numbers we’ve been working with:

There are also functions that take a value of one type and “convert” it to another:
3 Variables

A variable in Python is a name that refers to a value.

3.1 Variable assignment

Form of an assignment statement:

\[ \text{variable} = \text{expression} \]

How it’s executed
1. Evaluate the expression on the right-hand side.
2. Associate the result with the variable on the left-hand side.

Examples:

About Variable Assignment

• Python variables look like math variables.
  This could be Python or math:

  \[
  \begin{align*}
  p &= 5 \\
  q &= p \times 7 \\
  
  \end{align*}
  \]

  But they are very different!

Equality versus Assignment:

• In math, \( p = q + 10 \) states a fact about the value of \( p \) and of \( q + 10 \): that they are equal.

• In Python, \( p = q + 10 \) means something completely different, namely assignment.
  
  – Get the value variable \( q \) refers to.
  – Add 10 to that value.
  – Assign variable \( p \) the resulting value: the variable \( p \) is now a name for the resulting value, i.e., it refers to the resulting value.

• This is why you might reasonably say \( x = x + 1 \) (though it makes no sense in math!)

• We say \( x \) is assigned \( x + 1 \) or \( x \) gets \( x + 1 \)

• Programming languages usually have different symbols for assignment and equality. In Python, the symbol for equality is \( == \)

• In math, these are inconsistent:

  \[
  \begin{align*}
  p &= 5 \\
  q &= 7 \\
  p &= q + 10 \\
  
  \end{align*}
  \]

  \( p \) can’t be both 5 and 17!

• But in Python, it makes perfect sense. \( p \) starts out referring to 5, but then changes to refer to 17.
• You can change a variable’s value as many times as you want, and that may change its type too.

In mathematics, equations are descriptions that are simultaneously true. In Python, assignment statements look like equations but really specify a sequence of steps.

• Assignment is not symmetric

<table>
<thead>
<tr>
<th></th>
<th>math</th>
<th>Python</th>
</tr>
</thead>
<tbody>
<tr>
<td>result = a + b</td>
<td>legal (same meaning)</td>
<td>legal</td>
</tr>
<tr>
<td>a + b = result</td>
<td></td>
<td>illegal</td>
</tr>
</tbody>
</table>

• What does this do?

```python
x = 37
y = x + 2  # y is now 39
x = 20  # Is y now 22?
```

That is not how assignment works!

### 3.2 Naming variables

• These names are all perfectly all right in Python:

```python
x
my_average
result
```

• There are a few rules about variable names:
  - Must start with a letter (or underscore).
  - Can include letters, digits, and underscores, but nothing else.
  - Case matters:
    ```python
    age = 11
    aGe  # Error! This is not defined.
    ```

• In addition to rules, there are also conventions about choosing variable names:
  - We choose meaningful names for the sake of the humans who will read our code. Doing so is considered good programming style. For example, if you are computing the average of some numbers, “average” is better than “x”.
  - For names that include multiple words, we use “pothole case”: e.g., `average_grade`
  - We’ll get lots of practice choosing good names over the term.
4 Functions

4.1 Motivation

Q. Consider a cookbook with 12 cake recipes, 3 of them involving a buttercream frosting. We could repeat that part of the recipe 3 times. What is a better option?

A.

4.2 Defining a function

Give an example of defining a function in math:

Form of a Python function definition:

```
def function_name(parameters):
    body
```

def: a Python keyword indicating a function definition
function_name: the function name, always in pothole_case
parameters: 0 or more parameters, comma separated
body: 1 or more statements, often ending with a return statement;
      all body statements must be indented the same amount, usually 4 spaces

4.2.1 Return Statement

Form of a return statement:

```
return expression
```

return: a Python keyword
expression: an expression

How it’s executed:

- Evaluate the expression. This produces a value (which has a memory address).
- Exit the function and produce that value to the caller.
4.3 Calling a function

So far, we have defined what \( f \) is, but we haven’t used it. When we call a function, we ask Python to execute it (carry it out).

Form of a function call:

\[
\text{function name}(\text{arguments})
\]

Example function calls:

How a function call is executed:

1. Each argument is an expression. Evaluate these expressions, in order. These produce values. (Each value has a memory address.)
2. Store those memory addresses in the corresponding parameters.
3. Execute the body of the function.

5 Expressions vs. Statements

Example Python expressions:

Example Python statements:

The distinction in Python is complex. In this course, you can think of it as follows:

- Expressions can be reduced to a value.
- Statements are commands to do something.