

Introduction to Programming in Python

Arithmetic

Dr. Bill Young
Department of Computer Science
University of Texas at Austin

Last updated: June 4, 2021 at 11:04

Arithmetic Operations

Arithmetic is something you do frequently in computing. Here are some useful operations you can perform on numeric data types.

Name	Meaning	Example	Result
+	Addition	$34 + 1$	35
-	Subtraction	$34.0 - 0.1$	33.9
*	Multiplication	$300 * 30$	9000
/	Float division	$1 / 2$	0.5
//	Integer division	$1 // 2$	0
%	Remainder	$20 \% 3$	2
**	Exponentiation	$4 ** 0.5$	2.0

$(x \% y)$ is often referred to as “ $x \bmod y$ ”.

Some Arithmetic Expressions

```
>>> ans1 = 3 + (16 - 4) / 3
>>> ans2 = 13 % 2 - 1
>>> ans3 = 17 / 2
>>> ans4 = 17 // 2
>>> ans5 = 4 ** 2
>>> ans6 = 4 ** 0.5
>>> ans7 = 15 * 3
>>> ans8 = 15.0 * 3
```

What's in the 8 variables? How could you check if x is an even number?

Most arithmetic operations behave as you would expect for numeric data types.

- Combining two floats results in a float.
- Combining two ints results in an int (except for `/`). Use `//` for integer division.
- Dividing two ints gives a float. E.g., `5 / 2` yields 2.5.
- Combining a float with an int usually yields a float.

Python will figure out what the result should be and return a value of the appropriate data type.

Mixed Type Expressions

```
>>> 5 * 3 - 4 * 6      # (5 * 3) - (4 * 6)
-9
>>> 4.2 * 3 - 1.2
11.4000000000000002    # approximate result
>>> 5 // 2 + 4         # integer division
6
>>> 5 / 2 + 4          # float division
6.5
```

Notice operations on integers are usually *exact*, while operations on floats are *approximate*.

Augmented Assignment Statements

Python (like C) provides a shorthand syntax for some common assignments:

<code>i += j</code>	means the same as	<code>i = i + j</code>
<code>i -= j</code>	means the same as	<code>i = i - j</code>
<code>i *= j</code>	means the same as	<code>i = i * j</code>
<code>i /= j</code>	means the same as	<code>i = i / j</code>
<code>i //= j</code>	means the same as	<code>i = i // j</code>
<code>i %= j</code>	means the same as	<code>i = i % j</code>
<code>i **= j</code>	means the same as	<code>i = i ** j</code>

```
>>> x = 2.4
>>> x *= 3.7           # same as x = x * 3.7
>>> print(x)
8.88
```

Operator Precedence

Arithmetic expressions in Python attempt to match standard syntax. Thus,

$$3 + 4 * (5 + 2)$$

is interpreted as representing:

$$(3 + (4 * (5 + 2))).$$

That is, we perform the operation within parenthesis first, then the multiplication, and finally the addition.

To make this happen we need *precedence rules*.

Precedence

The following are the precedence rules for Python, with items higher in the chart having higher precedence.

Operator	Meaning
<code>+</code> , <code>-</code>	Unary plus, minus
<code>**</code>	Exponentiation
<code>not</code>	logical negation
<code>*</code> , <code>/</code> , <code>//</code> , <code>%</code>	Multiplication, division, integer division, remainder
<code>+</code> , <code>-</code>	Binary plus, minus
<code><</code> , <code><=</code> , <code>></code> , <code>>=</code>	Comparison
<code>==</code> , <code>!=</code>	Equal, not equal
<code>and</code>	Conjunction
<code>or</code>	Disjunction

Precedence Examples

```
>>> -3 * 4
-12
>>> - 3 + - 4
-7
>>> 3 + 2 ** 4
19
>>> 4 + 6 < 11 and 3 - 10 < 0
True
>>> 4 < 5 <= 17      # notice special syntax
True
>>> 4 + 5 < 2 + 7
False
>>> 4 + (5 < 2) + 7   # this surprised me!
11
```

Most of the time, the precedence follows what you would expect.

Use Parentheses to Override Precedence

Use parenthesis to override precedence or to make the evaluation clearer.

```
>>> 10 - 8 + 5          # an expression
7
>>> (10 - 8) + 5        # what precedence will do
7
>>> 10 - (8 + 5)         # override precedence
-3
>>> 5 - 3 * 4 / 2        # not particularly clear
-1.0
>>> 5 - ((3 * 4) / 2)    # much better
-1.0
```

Always try to make your code as easy to read as possible!

Exercise: Grade Computation

Suppose we want to solve the following problem:. Student Susie Q. has 3 exam grades and 2 project grades. Exams and projects have different possible point values. The exams together count 60% and the projects count 40%. Compute and print a grade report for this student.

Assume the following information:

Student: Susie Q.

Exam grades: 75/100, 85/90, 57/65

Project grades: 95/100, 150/200

Exercise: Grade Computation

Write a program to print a grade report that looks like this. Round each float to 2 decimal places (use the `round(value, 2)` function for that).

```
Grades for Susie Q.  
  Exam1: 75.0  
  Exam2: 94.44  
  Exam3: 87.69  
Exam average: 85.71  
  Proj1: 95.0  
  Proj2: 75.0  
Proj average: 85.0  
Course average: 85.43
```

Exercise Solution

In file Grade1.py:

```
student = "Susie Q."

# Normalize each exam score:
exam1Norm = (75 / 100) * 100.0
exam2Norm = (95 / 90) * 100.0
exam3Norm = (57 / 65) * 100.0

# Compute the average of the three exams:
examAvg = (exam1Norm + exam2Norm + exam3Norm) / 3

# Normalize each project score:
proj1Norm = (95 / 100) * 100.0
proj2Norm = (150 / 200) * 100.0

# Compute the average of the two projects:
projAvg = (proj1Norm + proj2Norm) / 2

# Find the weighted average:
courseAvg = examAvg * 0.6 + projAvg * 0.4

# Program continues on next slide
```

Exercise Solution

```
# Print the student's grade report:
print()
print("Grades for", student)
print("  Exam1:", round(exam1Norm, 2))
print("  Exam2:", round(exam2Norm, 2))
print("  Exam3:", round(exam3Norm, 2))
print("Exam average:", round(examAvg, 2))

print("  Proj1:", round(proj1Norm, 2))
print("  Proj2:", round(proj2Norm, 2))
print("Proj average:", round(projAvg, 2))

print("Course average:", round(courseAvg, 2))
```

Running the Program

```
> python Grade1.py  
  
Grades for Susie Q.  
  Exam1: 75.0  
  Exam2: 94.44  
  Exam3: 87.69  
Exam average: 85.71  
  Proj1: 95.0  
  Proj2: 75.0  
Proj average: 85.0  
Course average: 85.43
```