ARITHMETIC AND COMPARISON OPERATORS

INTRODUCTION

This chapter covers the various built-in operators, which Python has to offer.

OPERATORS

These operations (operators) can be applied to all numeric types:

Operator	Description	Example
+, -	Addition, Subtraction	10 -3
*, %	Multiplication, Modulo	27 % 7 Result: 6

/	Division
	This operation results in different results for Python 2.x (like floor division) and Python 3.x
		Python3:
		`>>> 10  / 3`
		`3.3333333333333335`
		and in Python 2.x:
		`>>> 10  / 3`
		`3`

//	Truncation Division (also known as floordivision or floor division)
	The result of this division is the integral part of the result, i.e. the fractional part is truncated, if there is any.
	It works both for integers and floating-point numbers, but there is a difference in the type of the results: If both the divident and the divisor are integers, the result will be also an integer. If either the divident or the divisor is a float, the result will be the truncated result as a float.
		`>>> 10 // 3`
		`3`
		If at least one of the operands is a float value, we get a truncated float value as the result.
		`>>> 10.0 // 3`
		`3.0`
		`>>>`
		A note about efficiency:
		The results of int(10 / 3) and 10 // 3 are equal. But the "//" division is more than two times as fast! You can see this here:
		`In [9]: %%timeit`
		`for x in range(1,`

		100):
		    y = int(100 / x)
		   ...:
		100000 loops, best of 3: 11.1 µs per loop
		
		In [10]: %%timeit
		for x in range(1, 100):
		    y = 100 // x
		   ....:
		100000 loops, best of 3: 4.48 µs per loop

+x, -x	Unary minus and Unary plus (Algebraic signs)	-3
~x	Bitwise negation	~3 - 4 Result: -8
**	Exponentiation	10 ** 3 Result: 1000
or, and, not	Boolean Or, Boolean And, Boolean Not	(a or b) and c
in	"Element of"	1 in [3, 2, 1]
<, <=, >, >=, !=, ==	The usual comparison operators	2 <= 3
|, &, ^	Bitwise Or, Bitwise And, Bitwise XOR	6 ^ 3
<<, >>	Shift Operators	6 << 3