🧭 Python Operatoren – Der Ultimative Guide

Python hat viele Arten von Operatoren. Hier lernst du die wichtigsten Gruppen:

🧮 1. Arithmetische Operatoren

Diese Operatoren werden verwendet, um **mathematische Berechnungen** durchzuführen.

Operator	Bedeutung	Beispiel	Ergebnis
`+`	Addition	`5 + 3`	`8`
`-`	Subtraktion	`10 - 6`	`4`
`*`	Multiplikation	`4 * 2`	`8`
`/`	Division	`8 / 2`	`4.0`
`//`	Ganzzahlige Division	`7 // 3`	`2`
`%`	Modulo (Restwert)	`10 % 3`	`1`
`**`	Potenzierung	`2 ** 3`	`8`

Beispiel:

```python
a = 9
b = 4
print(a + b)   # 13
print(a / b)   # 2.25
print(a // b)  # 2
print(a % b)   # 1
print(a ** b)  # 6561
```

⚖️ 2. Vergleichsoperatoren (Relationale Operatoren)

Vergleichsoperatoren prüfen **Beziehungen zwischen Werten** und liefern immer einen **booleschen Wert** (`True` oder `False`).

Operator	Bedeutung	Beispiel	Ergebnis
`==`	Gleich	`5 == 5`	`True`
`!=`	Ungleich	`3 != 4`	`True`
`>`	Größer als	`7 > 2`	`True`
`<`	Kleiner als	`2 < 5`	`True`
`>=`	Größer oder gleich	`5 >= 5`	`True`
`<=`	Kleiner oder gleich	`4 <= 3`	`False`

Beispiel:

```python
x = 10
y = 20
print(x == y)  # False
print(x < y)   # True
print(x != y)  # True
```

🔗 3. Logische Operatoren

Logische Operatoren kombinieren mehrere **boolesche Ausdrücke**.

Operator	Bedeutung	Beispiel	Ergebnis
`and`	Wahr, wenn **beide** Wahr sind	`True and False`	`False`
`or`	Wahr, wenn **mindestens einer** Wahr ist	`True or False`	`True`
`not`	Negiert den Wert	`not True`	`False`

Beispiel:

```python
x = 5
print(x > 3 and x < 10)  # True
print(x > 10 or x == 5)  # True
print(not(x > 3))        # False
```

💾 4. Zuweisungsoperatoren

Diese Operatoren dienen dazu, **Werte Variablen zuzuweisen** – oft mit einer Berechnung kombiniert.

Operator	Bedeutung	Beispiel	Entspricht
`=`	Zuweisung	`x = 5`	`x = 5`
`+=`	Addiere und weise zu	`x += 3`	`x = x + 3`
`-=`	Subtrahiere und weise zu	`x -= 2`	`x = x - 2`
`*=`	Multipliziere und weise zu	`x *= 4`	`x = x * 4`
`/=`	Dividiere und weise zu	`x /= 2`	`x = x / 2`
`//=`	Ganzzahldivision und Zuweisung	`x //= 3`	`x = x // 3`
`%=`	Modulo und Zuweisung	`x %= 2`	`x = x % 2`
`**=`	Potenzierung und Zuweisung	`x **= 2`	`x = x ** 2`

Beispiel:

```python
x = 10
x += 5
print(x)  # 15
x **= 2
print(x)  # 225
```

Python 3.14 Operatoren - Ultimative Tutorial

Michael Schrot | YouTube.com/@mschrot | Folge 50

🔢 5. Bitweise Operatoren

Diese Operatoren arbeiten **auf Bit-Ebene** – sie sind nützlich für Low-Level-Programmierung.

Operator	Bedeutung	Beispiel	Erklärung
`&`	Bitweises UND	`5 & 3`	0101 & 0011 → 0001
`|`	Bitweises ODER	`5 | 3`	0101 | 0011 → 0111
`^`	Bitweises XOR	`5 ^ 3`	0101 ^ 0011 → 0110
`~`	Bitweises NOT	`~5`	Invertiert alle Bits
`<<`	Linksverschiebung	`5 << 1`	0101 → 1010 (10)
`>>`	Rechtsverschiebung	`5 >> 1`	0101 → 0010 (2)

Beispiel:

```python
a = 5  # 0101
b = 3  # 0011
print(a & b)  # 1
print(a | b)  # 7
print(a ^ b)  # 6
print(a << 1) # 10
```

🧩 6. Identitätsoperatoren

Diese Operatoren prüfen, ob zwei Variablen **auf dasselbe Objekt im Speicher** zeigen.

Operator	Bedeutung	Beispiel	Ergebnis
`is`	Ist dasselbe Objekt	`a is b`	`True` oder `False`
`is not`	Ist **nicht** dasselbe Objekt	`a is not b`	`True` oder `False`

Beispiel:

```python
x = ["Apfel", "Banane"]
y = ["Apfel", "Banane"]
z = x

print(x is z)      # True (gleiche Referenz)
print(x is y)      # False (andere Objekte)
print(x == y)      # True (gleicher Inhalt)
```

📦 7. Mitgliedsoperatoren

Diese Operatoren prüfen, ob ein Wert in einer **Sequenz** (z. B. Liste, String, Tuple) vorkommt.

Operator	Bedeutung	Beispiel	Ergebnis
`in`	Wert ist vorhanden	`"a" in "Haus"`	`True`
`not in`	Wert ist **nicht** vorhanden	`"z" not in "Haus"`	`True`

Beispiel:

```python
fruits = ["Apfel", "Banane", "Kirsche"]
print("Banane" in fruits)     # True
print("Birne" not in fruits)  # True
```

⚙️ 8. Ternärer Operator (Kurzform von if-else)

Der **ternäre Operator** erlaubt eine **kurze if-else-Schreibweise**.

Syntax:

```python
x if condition else y
```

Beispiel:

```python
age = 18
status = "Volljährig" if age >= 18 else "Minderjährig"
print(status)  # Volljährig
```

🧠 9. Operator-Priorität (Reihenfolge der Auswertung)

Python folgt einer bestimmten **Rangordnung**, welche Operatoren zuerst ausgewertet werden:

1. `()` – Klammern
2. `**` – Potenzierung
3. `+x`, `-x`, `~x` – unäre Operatoren
4. `*`, `/`, `//`, `%`
5. `+`, `-`
6. `<<`, `>>`
7. `&`, `^`, `|`
8. Vergleichsoperatoren (`<`, `>`, `==`, `!=`, etc.)
9. `not`, `and`, `or`
10. `=` und andere Zuweisungen

📑 Zusammenfassung

Kategorie	Beispiele	Hauptzweck
Arithmetisch	`+`, `-`, `*`, `/`, `**`	Rechnen
Vergleich	`==`, `<`, `>=`	Werte prüfen
Logisch	`and`, `or`, `not`	Bedingungen kombinieren
Zuweisung	`=`, `+=`, `*=`	Variablen manipulieren
Bitweise	`&`, `|`, `^`, `<<`, `>>`	Bits manipulieren
Identität	`is`, `is not`	Objekte vergleichen
Mitgliedschaft	`in`, `not in`	Listen, Strings prüfen