PCEP-30-02 1.4 – Wählen geeigneter Operatoren und Datentypen für das Problem

Lernziele (PCEP 1.4)

- Nutzung von numerischen Operatoren (**, *, /, %, //, +, -)
- Anwendung von String-Operatoren (*, +)
- Einsatz von Zuweisungs- und Kurzschreiboperatoren
- Verständnis von unären und binären Operatoren

- Kenntnis von Operator-Prioritäten und Bindung
- Nutzung von bitweisen Operatoren (`~`, `&`, `^`, `|`, `<<`, `>>`)
- Einsatz von Boolean-Operatoren (`not`, `and`, `or`) und das Erstellen von Boolean-Ausdrücken
- Anwendung von Vergleichsoperatoren (`==`, `!=`, `>`, `>=`, `<`, `<=`)

Übersicht der Themen

1. Numerische Operatoren

2. String-Operatoren

3. Zuweisungs- und Kurzschreiboperatoren

4. Unäre und binäre Operatoren

5. Prioritäten und Bindung
6. Bitweise Operatoren
7. Boolean-Operatoren & Ausdrücke
8. Vergleichsoperatoren
9. Genauigkeit von Floating-Point-Zahlen
10. Typumwandlung

1. Numerische Operatoren

Numerische Operatoren ermöglichen mathematische Berechnungen, z. B.:

- **Exponentiation: ****
- **Multiplikation: ***
- **Division: /**
- **Modulo: %**
- **Ganzzahlige Division: //**

Alltagsbeispiel:
Berechne den Gesamtpreis einer Bestellung, indem du die Einzelpreise addierst oder Rabatte abziehst.

Beispiel: Numerische Operatoren (Code Example 0)

```
a = 10
b = 3
print("Exponentiation (10 ** 3):", a ** 3)
print("Multiplikation (10 * 3):", a * b)
print("Division (10 / 3):", a / b)
print("Modulo (10 % 3):", a % b)
print("Ganzzahlige Division (10 // 3):", a // b)
print("Addition (10 + 3):", a + b)
print("Subtraktion (10 - 3):", a - b)
```

2. String-Operatoren

String-Operatoren ermöglichen:

- **Konkatenation:** Verbinden von Strings mit +
- **Wiederholung:** Wiederholen von Strings mit *

Alltagsbeispiel:
Erstelle eine Begrüßungsnachricht, indem du Vor- und Nachname zusammenfügst oder einen Titel mehrfach wiederholst.

Beispiel: String-Operatoren (Code Example 1)

```
greeting = "Hello, "
name = "Alice"
print(greeting + name)          # Ausgabe: Hello, Alice
print(name * 3)                 # Ausgabe: AliceAliceAlice
```

3. Zuweisungs- und Kurzschreiboperatoren

Neben der einfachen Zuweisung (=) gibt es Kurzschreiboperatoren, die Berechnung und Zuweisung kombinieren, z. B.:

- `+=`, `-=`, `*=`, `/=`, `%=` und mehr.

Alltagsbeispiel:
Erhöhe einen Zähler bei jedem Klick oder reduziere einen Preis mit einem Rabatt.

Beispiel: Zuweisungs-Operatoren (Code Example 2)

```
count = 5
count += 3   # entspricht: count = count + 3
print("Neuer Wert:", count)  # Ausgabe: 8
```

4. Unäre und binäre Operatoren

- **Unäre Operatoren:** Wirken auf einen einzelnen Operanden, z. B. `-x` oder `not x`.

- **Binäre Operatoren:** Wirken auf zwei Operanden, z. B. `x + y`.

Alltagsbeispiel:
Negiere einen Wert (z. B. Temperaturdifferenz) oder addiere zwei Zahlen.

Beispiel: Unäre und Binäre Operatoren (Code Example 3)

```
x = 10
print("Negation von x:", -x)        # Ausgabe: -10
print("Addition (x + 5):", x + 5)      # Ausgabe: 15
print("Boolean Negation (not True):", not True)  # Ausgabe: False
```

5. Prioritäten und Bindung

Operatoren haben unterschiedliche Prioritäten.
Beispiel:

- `2 + 3 * 4` wird als `2 + (3 * 4)` ausgewertet.

- Klammern können die Reihenfolge beeinflussen.

Alltagsbeispiel:
Berechne zuerst den Steuerbetrag, bevor du den Endpreis addierst.

Beispiel: Operator-Priorität (Code Example 4)

```
result = 2 + 3 * 4  # entspricht 2 + (3 * 4) = 14
print("Ergebnis:", result)
result_with_parentheses = (2 + 3) * 4  # entspricht 5 * 4 = 20
print("Ergebnis mit Klammern:", result_with_parentheses)
```

6. Bitweise Operatoren

Bitweise Operatoren arbeiten auf der Bit-Ebene von Ganzzahlen:

- **NOT: ~**
- **AND: &**
- **XOR: ^**
- **OR: |**
- **Linksverschiebung: <<**

Alltagsbeispiel:
Bitweise Operationen werden zur Verarbeitung von Flags oder Berechtigungen genutzt.

Beispiel: Bitweise Operatoren (Code Example 5)

```
a = 12       # binär: 1100
b = 5        # binär: 0101
print("Bitweises AND:", a & b)    # 1100 & 0101 = 0100 (4)
print("Bitweises OR:", a | b)     # 1100 | 0101 = 1101 (13)
print("Bitweises XOR:", a ^ b)    # 1100 ^ 0101 = 1001 (9)
print("Bitweise NOT (~a):", ~a)   # Negiert die Bits von a
print("Linksverschiebung (a << 1):", a << 1)  # 12 << 1 = 24
print("Rechtsverschiebung (a >> 1):", a >> 1) # 12 >> 1 = 6
```

7. Boolean-Operatoren und Boolean-Ausdrücke

Boolean-Operatoren:

- `not`, `and`, `or`

Boolean-Ausdrücke kombinieren Wahrheitswerte, um Bedingungen zu prüfen.
Alltagsbeispiel:
Entscheide, ob du basierend auf dem Wetter und der Temperatur draußen spazieren gehst.

Beispiel: Boolean-Operatoren (Code Example 6)

```
is_raining = False
is_warm = True
print("Gehe spazieren:", is_warm and not is_raining)
```

8. Vergleichsoperatoren (Relationale Operatoren)

Vergleichsoperatoren prüfen Beziehungen zwischen Werten:

- ==, !=, >, >=, <, <=

Alltagsbeispiel:
Überprüfe, ob dein Kontostand hoch genug ist, um einen Kauf zu tätigen.

Beispiel: Relationale Operatoren (Code Example 7)

```
a = 10
b = 20
print("a == b:", a == b)
print("a != b:", a != b)
print("a < b:", a < b)
print("a <= 10:", a <= 10)
```

9. Genauigkeit von Floating-Point-Zahlen

Floating-Point-Zahlen können aufgrund ihrer binären Darstellung ungenau sein.
Beispiel:
`0.1 + 0.2` ergibt oft nicht exakt `0.3`.

Alltagsbeispiel:
Bei finanziellen Berechnungen ist Vorsicht geboten – manchmal empfiehlt sich die Verwendung von Dezimaltypen.

Beispiel: Floating-Point Genauigkeit (Code Example 8)

```
result = 0.1 + 0.2
print("0.1 + 0.2 =", result)  # Ausgabe: 0.30000000000000004
```

10. Typumwandlung (Type Casting)

Typumwandlung wird verwendet, um Werte zwischen Datentypen zu konvertieren:

- Beispiele: `int()`, `float()`, `str()`

Alltagsbeispiel:
Konvertiere Benutzereingaben (die als Strings vorliegen) in Zahlen, um mathematische Operationen durchzuführen.

Beispiel: Typumwandlung (Code Example 9)

```
num_str = "123"
num_int = int(num_str)
print("Typumwandlung:", num_int + 1)   # Ausgabe: 124
```

Challenge Slides

Challenge 1: Kombinierte Operationen

Erstelle einen Ausdruck, der:

- Eine Zahl in wissenschaftlicher Notation verwendet,
- Eine String-Konkatenation durchführt und
- Boolean-Operatoren kombiniert.

Diskutiere, wie unterschiedliche Operatoren zusammenwirken.

Challenge 2: Bitweise Logik im Alltag

Erkläre in eigenen Worten, wie bitweise Operatoren z. B. bei Dateiberechtigungen oder Flag-Variablen angewendet werden können.

Challenge 3: PEP-8 Check

Schreibe einen kurzen Python-Code, der mehrere Variablen und Ausdrücke enthält. Achte darauf, dass der Code den PEP-8 Empfehlungen entspricht (Namenskonventionen, Einrückung, Zeilenlänge).

Zusammenfassung (PCEP 1.4)

- **Numerische Operatoren:** Ermöglichen mathematische Berechnungen.
- **String-Operatoren:** Dienen der Verkettung und Wiederholung von Text.
- **Zuweisungs-/Kurzschreiboperatoren:** Vereinfachen die Manipulation von Variablen.
- **Unäre/Binäre Operatoren:** Arbeiten auf einem

- **Bitweise Operatoren:** Arbeiten direkt auf der Binärebene.
- **Boolean-Operatoren & Relationale Operatoren:** Ermöglichen logische Entscheidungen und Vergleiche.
- **Floating-Point Genauigkeit:** Erfordert besondere Vorsicht bei Berechnungen.
- **Typumwandlung:** Konvertiert Werte in den gewünschten Datentyp.