HELENE-WEBER-SCHULE BUCHEN	Algorithmen und Datenstrukturen **Python: Befehlsübersicht**	**Name:** **Datum:** **Fach:** Informatik

1. Variablen

- Variablen speichern Werte.
- Variablen können jederzeit überschrieben werden.
- In Python müssen keinen Datentyp angeben. Der Datentyp wird automatisch zugewiesen.

```
1  name = "Alice"   # String
2  alter = 25       # Integer
3  preis = 19.99    # Float
4  ist_aktiv = True  # Boolean
```

2. Datentypen

- Integer (Ganzzahlen): **int**
- Float (Gleitkommazahlen): **float**
- String (Zeichenkette): **str**
- Boolean (Wahrheitswerte): **bool**

```
1  name = "Alice"   # String
2  alter = 25       # Integer
3  preis = 19.99    # Float
4  ist_aktiv = True  # Boolean
```

Datentyp ermitteln:

- **type():** Mit der Funktion type() wird der Datentyp ermittelt.

```
1  type(variable)
2
3  # Beispiel
4  x = "Text123"
5  print(type(x))
```

```
Kommandozeile
>>> %Run test.py
  <class 'str'>
>>>
```

3. Benutzereingabe

- **input():** liest Eingaben vom Benutzer als String ein.

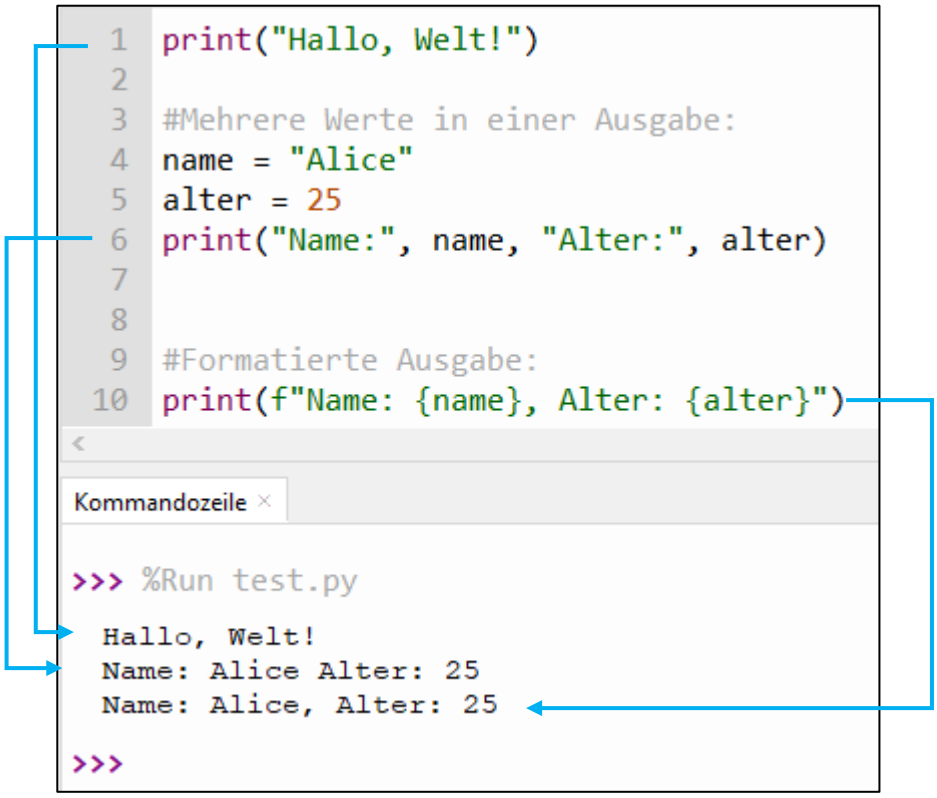
```
1  name = input("Gib deinen Namen ein: ")
```

- Wenn Sie eine Zahl benötigen, müssen Sie die Eingabe konvertieren:

```
1  alter = int(input("Gib dein Alter ein: "))  # für ganze Zahlen
2  preis = float(input("Gib den Preis ein: "))  # für Gleitkommazahlen
```

4. Ausgabe

- **print():** gibt Daten auf dem Bildschirm bzw. im Terminal / Kommandozeile aus.



5. Operatoren

Arithmetische Operatoren

- Addition: **+**
- Subtraktion: **-**
- Multiplikation: *****
- Division: **/** (Ergebnis immer float)
- Ganzzahl-Division: **//**
- Modulo (Rest-Division): **%**
- Potenz: ******

```
a = 5
b = 2
print(a + b)  # Ausgabe: 7
print(a - b)  # Ausgabe: 3
print(a * b)  # Ausgabe: 10
print(a / b)  # Ausgabe: 2.5
print(a % b)  # Ausgabe: 25
print(a // b) # Ausgabe: 2
print(a ** b) # Ausgabe: 25
```

Vergleichsoperatoren (Ergebnis = bool)

Vergleichsoperatoren geben einen booleschen Wert zurück (True oder False), abhängig davon, ob die Bedingung erfüllt ist.

- Gleich: **==**
- Ungleich: **!=**
- Größer: **>**
- Kleiner: **<**
- Größer oder gleich: **>=**
- Kleiner oder gleich: **<=**

```
a = 10
b = 10
print(a == b)  # Ausgabe: True
print(a >= b)  # Ausgabe: True
print(a <= b)  # Ausgabe: True


a = 10
b = 5
print(a != b)  # Ausgabe: True
print(a > b)   # Ausgabe: True
print(a < b)   # Ausgabe: False
print(a >= b)  # Ausgabe: True
print(a <= b)  # Ausgabe: False
```

Logische Operatoren

Logische Operatoren werden verwendet, um mehrere Bedingungen zu kombinieren.
Sie geben ebenfalls einen booleschen Wert zurück.

- Und: **and**
- Oder: **or**
- Nicht: **not**

```
a = 10
b = 20
print(a > 5 and b > 15)  # Ausgabe: True, weil beide Bedingungen erfüllt sind
print(a > 15 and b > 15)  # Ausgabe: False, weil a > 15 falsch ist

a = 10
b = 20
print(a > 15 or b > 15)  # Ausgabe: True, weil b > 15 wahr ist
print(a > 15 or b < 15)  # Ausgabe: False, weil beide Bedingungen falsch sind

a = True
b = False
print(not a)  # Ausgabe: False, weil a True ist
print(not b)  # Ausgabe: True, weil b False ist
```

6. Verzweigung (Bedingungen) → if, elif, else

if (Bedingung):

Anweisung1

elif (weitere Bedingung):

Anweisung2

else: für alle anderen Fälle

Anweisung3

Auf die Einrückung achten!

```
1  zahl = 10
2
3  if zahl > 0:
4      print("Die Zahl ist positiv")
5  elif zahl == 0:
6      print("Die Zahl ist Null")
7  else:
8      print("Die Zahl ist negativ")
```

Kommandozeile

```
>>> %Run test.py
  Die Zahl ist positiv
```

7. Schleifen

while-Schleife

while-Schleife: Wiederholt Code, solange eine Bedingung wahr ist.

Beispiel
Die Variable wird in jedem Schleifendurchlauf um den Wert 1 erhöht, bis die Bedingung x < 5 nicht mehr erfüllt ist.

→ Hierbei wird die Variable in jedem Schleifendurchlauf ausgegeben.

```
1  x = 0
2  while x < 5:
3      print(x)
4      x += 1  # Erhöhe x um 1
5
```

Kommandozeile

```
>>> %Run test.py
  0
  1
  2
  3
  4
```

for-Schleife

for-Schleife: Iteration über eine feste Anzahl von Schritten z.B. mit range() oder einer Liste (Array)

range(): Erzeugt eine Sequenz von Zahlen → **Syntax: range*(start, stop, step)***

- *Mit **start** können Sie den Anfangswert einstellen. start auch weglassen (optional). Dann fängt range() bei 0 an zu zählen.*
- *Mit **stop** legen Sie den Endwert fest. Dieser Wert muss immer angeben werden.*
  Aber aufgepasst: range() hört bei stop-1 auf zu zählen!
- *Mit step können Sie die Schrittweite angeben. step kann auch weglassen werden (optional).*

```
1  for i in range(5):  # 0 bis 4
2      print(i)
```

Kommandozeile

```
>>> %Run test.py
0
1
2
3
4
```

```
1  berufe = ["Arzt", "Bäcker", "Chemikant", "Dachdecker"]
2  for i in berufe:
3      print(i)
```

Kommandozeile

```
>>> %Run test.py
Arzt
Bäcker
Chemikant
Dachdecker
```

break und **continue**

- **break**: Beendet die Schleife vorzeitig.
- **continue**: Überspringt den aktuellen Schleifendurchlauf.

```
1  for i in range(5):
2      if i == 3:
3          break  # Schleife wird bei 3 beendet
4      print(i)
5
6  print("----------------------------")
7
8  for i in range(5):
9      if i == 2:
10         continue  # 2 wird übersprungen
11     print(i)
```

Kommandozeile

```
>>> %Run test.py
0
1
2
----------------------------
0
1
3
4
```