# Funktionen & Rückgabewerte

Python 3 · kompakte Grundlagen mit direkt nutzbaren Beispielen



## 1 | Funktion definieren

- def legt eine Funktion an.
- Aufruf erfolgt mit Klammern.

```
def begruessen():
    print("Hallo!")

begruessen()
```

## 2 | Parameter

- Parameter nehmen Eingaben entgegen.
- Namen sollen Bedeutung und Einheit zeigen.

```
def flaeche(breite_mm, hoehe_mm):
    return breite_mm * hoehe_mm

a = flaeche(40, 25)
```

## 3 | return

- return gibt Wert zurück und beendet Funktion.
- Ohne return ist Ergebnis None.

```
def quadrat(x):
    return x * x

ergebnis = quadrat(5)
```

## 4 | Standardwerte

- Optionale Parameter erhalten Default.
- Pflichtparameter stehen zuerst.

```
def ausgabe(text, einheit="mm"):
    print(f"{text} {einheit}")

ausgabe(25)
```

## 5 | Benannte Argumente

- Erhöhen Lesbarkeit.
- Reihenfolge ist bei benannten Argumenten frei.

```
volumen = quader(
    breite=40,
    tiefe=20,
    hoehe=10
)
```

## 6 | Mehrere Rückgabewerte

- Python gibt praktisch ein Tupel zurück.
- Direktes Entpacken ist üblich.

```
def min_max(werte):
    return min(werte), max(werte)

klein, gross = min_max([3, 8, 1])
```

# Funktionen & Rückgabewerte

Python 3 · kompakte Grundlagen mit direkt nutzbaren Beispielen



## 7 | Lokale Variablen

- In Funktion angelegte Namen sind lokal.
- Globale Werte besser als Parameter übergeben.

```
faktor = 2
def rechnen(wert):
    ergebnis = wert * faktor
    return ergebnis
```

## 8 | Docstrings

- Beschreiben Zweck, Parameter und Rückgabe.
- Stehen direkt nach def.

```
def umfang(radius):
    """Berechnet Kreisumfang in gleicher Einheit."""
    return 2 * 3.14159 * radius
```

## 9 | Kleine Funktionen

- Eine Aufgabe pro Funktion.
- Aussagekräftiger Name statt langer Kommentar.
- Eingaben statt versteckter globaler Zustände.
- Rückgabewert statt unnötiger print-Ausgabe.
- Randfälle gezielt testen.

## 10 | Typen andeuten

- Type Hints dokumentieren erwartete Typen.
- Sie erzwingen Typen zur Laufzeit nicht.

```
def kosten(masse_g: float, preis_kg: float) -> float:
    return masse_g / 1000 * preis_kg
```

## 11 | Typische Fehler

| Problem | Prüfen |
|---|---|
| None erhalten | return fehlt |
| NameError | lokaler Bereich |
| Argument fehlt | Signatur/Aufruf |
| falsche Reihenfolge | benannte Argumente |
| zu lange Funktion | aufteilen |

## 12 | Mini-Übung: Umrechnen

**Aufgabe**

Schreibe eine Funktion von Millimeter nach Meter.

**Musterlösung**

```
def mm_in_m(wert_mm):
    return wert_mm / 1000

print(mm_in_m(250))
```