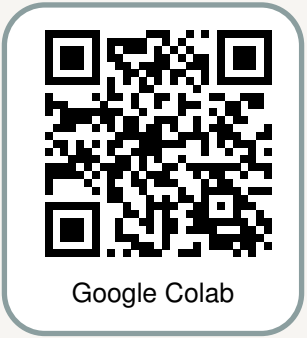
Python Grundlagen

Funktionen in Python


Google Colab

Lernziele

Am Ende dieser Lektion können Sie:

- ✓ Funktionen definieren und aufrufen
- ✓ Parameter und Rückgabewerte nutzen
- ✓ Den Unterschied zwischen `print` und `return` verstehen

Was ist eine Funktion?

Funktion = Wiederverwendbarer Code-Baustein

- Organisiert Code in logische Einheiten
- Vermeidet Wiederholungen
- Macht Programme übersichtlicher
- Wie ein Rezept: einmal definieren, oft verwenden

Syntax: Funktion definieren

Syntax: `def funktionsname(parameter):`

```
1 def funktionsname(parameter1, parameter2):
2     # Code-Block der Funktion
3     ergebnis = parameter1 + parameter2
4     return ergebnis
```

- `def`: Leitet Funktionsdefinition ein
- Parameter: Eingabewerte (optional)
- `return`: Gibt Ergebnis zurück (optional)

Beispiel: Einfache Begrüßung

Definition der Funktion:

```
def begruessung():
    print("Hallo, wie geht es dir heute?")
```

Aufruf der Funktion:

```
begruessung()
```

Ausgabe:

```
Hallo, wie geht es dir heute?
```

Funktionen mit Parametern

```
1 def personalisierte_begruessung(name):
2     print(f"Hallo {name}, wie geht es dir?")
```

Verschiedene Aufrufe:

```
personalisierte_begruessung("Aaron")
# Ausgabe: Hallo Aaron, wie geht es dir?

personalisierte_begruessung("Maria")
# Ausgabe: Hallo Maria, wie geht es dir?
```

Mehrere Parameter

```
def addiere(a, b):
    ergebnis = a + b
    print(ergebnis)

addiere(4, 5)  # Ausgabe: 9
```

```
def volumen_rechteck(l, b, h):
    volumen = l * b * h
    return volumen

v = volumen_rechteck(5, 4, 3)
print(v)  # Ausgabe: 60
```

Wichtig: `return` vs. `print` - Teil 1

Mit `print`:

```
1 def addiere_print(a, b):
2     ergebnis = a + b
3     print(ergebnis)
4
5 x = addiere_print(3, 4)   # Ausgabe: 7
6 print(x)                  # Ausgabe: None
```

`print` gibt **nichts zurück**!
Die Variable `x` enthält `None`.

Wichtig: `return` vs. `print` - Teil 2

Mit `return`:

```
def addiere_return(a, b):
    ergebnis = a + b
    return ergebnis

y = addiere_return(3, 4)  # Keine Ausgabe
print(y)                  # Ausgabe: 7
print(y * 2)              # Ausgabe: 14
```

`return` gibt den Wert zurück!
Die Variable `y` enthält die Zahl `7` und kann weiterverwendet werden.

Der Unterschied

`print` zeigt nur an
`return` gibt einen Wert zurück, der weiterverwendet werden kann

```
x = addiere_print(3, 4)
print(x * 2)   # TypeError: unsupported operand type(s
   )
               # for *: 'NoneType' and 'int'
```

Benannte Parameter

```
def beschreibe_person(name, alter, stadt):
    print(f"{name} ist {alter} Jahre alt und lebt in {stadt}.")
```

```
# Positionsbasiert
beschreibe_person("Lisa", 25, "Berlin")

# Mit benannten Parametern (Reihenfolge egal!)
beschreibe_person(stadt="Muenchen", name="Tom", alter=30)
```

Standardparameter (Default-Werte)

Parameter können Standardwerte haben:

```
1 def addition(a, b=3):
2     return a + b
```

```
print(addition(5))       # Ausgabe: 8 (5 + 3)
print(addition(5, 10))   # Ausgabe: 15 (5 + 10)
```

Wenn kein Wert für `b` angegeben wird, wird der Standardwert `3` verwendet.

Funktionen + Kontrollstrukturen

Funktionen können if-Statements und Schleifen enthalten:

```
def pruefe_zahl(zahl):
    if zahl > 0:
        return "Die Zahl ist positiv."
    elif zahl < 0:
        return "Die Zahl ist negativ."
    else:
        return "Die Zahl ist null."
```

```
for i in range(-2, 3):
    print(f"Fuer die Zahl {i}: {pruefe_zahl(i)}")
```

Funktionen rufen Funktionen auf

```python
def eingabe_verarbeiten():
    name = input("Wie heisst du? ")
    return name

def begruessung_anzeigen(person):
    print(f"Herzlich willkommen, {person}!")

def hauptprogramm():
    benutzer = eingabe_verarbeiten()
    begruessung_anzeigen(benutzer)

# Programmstart
hauptprogramm()
```

Praktisches Beispiel: Rechner

```
def berechne(a, b, operation="addiere"):
    if operation == "addiere":
        return a + b
    elif operation == "subtrahiere":
        return a - b
    elif operation == "multipliziere":
        return a * b
    else:
        return None
```

```
print(berechne(5, 3))                          # 8
print(berechne(5, 3, "multipliziere"))         # 15
```

Mini-Übung 1

Schreibe eine Funktion `verdopple(zahl)`, die:

1. Eine Zahl als Parameter nimmt
2. Die Zahl mit 2 multipliziert
3. Das Ergebnis zurückgibt (nicht ausgibt!)

Teste die Funktion mit `verdopple(5)`

Lösung: Mini-Übung 1

```
def verdopple(zahl):
    return zahl * 2

ergebnis = verdopple(5)
print(ergebnis)  # 10
```

Mini-Übung 2

Schreibe eine Funktion `ist_gerade(zahl)`, die:

1. Prüft, ob die Zahl gerade ist
2. `True` zurückgibt, wenn gerade
3. `False` zurückgibt, wenn ungerade

Tipp: Nutze den Modulo-Operator %

Lösung: Mini-Übung 2

```
def ist_gerade(zahl):
    if zahl % 2 == 0:
        return True
    else:
        return False

# Oder kuerzer:
def ist_gerade(zahl):
    return zahl % 2 == 0

print(ist_gerade(4))    # True
print(ist_gerade(7))    # False
```

Übung: Funktionen kombinieren

Schreibe ein kleines Programm mit drei Funktionen:

1. `hole_zahlen()`: Fragt nach zwei Zahlen mit `input()`
2. `multipliziere(a, b)`: Multipliziert zwei Zahlen
3. `zeige_ergebnis(zahl)`: Gibt das Ergebnis aus

Rufe alle drei Funktionen nacheinander auf!

Lösung - Teil 1: Funktionen definieren

```
def hole_zahlen():
    a = int(input("Erste Zahl: "))
    b = int(input("Zweite Zahl: "))
    return a, b

def multipliziere(a, b):
    return a * b

def zeige_ergebnis(zahl):
    print(f"Das Ergebnis ist: {zahl}")
```

Lösung - Teil 2: Hauptprogramm

```
# Hauptprogramm
a, b = hole_zahlen()
ergebnis = multipliziere(a, b)
zeige_ergebnis(ergebnis)
```

Die drei Funktionen werden nacheinander aufgerufen:
1. Zahlen eingeben $\rightarrow$ 2. Multiplizieren $\rightarrow$ 3. Ergebnis anzeigen

Gute Funktionen sind:

- **Beschreibend benannt**: `berechne_durchschnitt()` statt `func1()`
- **Fokussiert**: Eine Funktion = eine Aufgabe
- **Wiederverwendbar**: Allgemein formuliert
- **Dokumentiert**: Kommentare erklären den Zweck

Vermeide: Zu viele Parameter (>3-4), unklare Namen, Funktionen die zu viel auf einmal machen

Zusammenfassung

- Funktionen definieren mit `def`
- Parameter für flexible Funktionen
- `return` gibt Werte zurück (wichtig!)
- Standardwerte für Parameter möglich
- Funktionen können andere aufrufen

Hausaufgaben

1. Schreibe eine Funktion `berechne_summe(liste)`, die alle Zahlen in einer Liste addiert
2. Erstelle eine Funktion `finde_maximum(a, b, c)`, die die größte von drei Zahlen zurückgibt
3. Schreibe eine Funktion `ist_palindrom(wort)`, die prüft, ob ein Wort vorwärts und rückwärts gleich ist