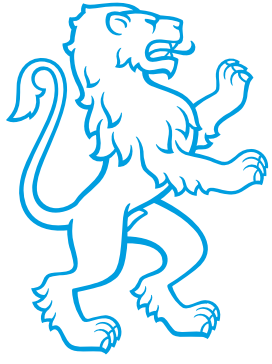


Kantonsschule Im Lee

Informatik: Programmieren

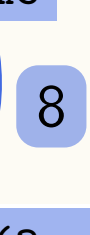
👩‍🍳 Kapitel 4: **Funktionen mit `return`**: Mehrere Funktionen

Funktionen: Mit `return`

```
def summiere(x1, x2):
    summe = x1 + x2
    return summe



res = summiere(3, 5)

print(res)
```

Funktionen: Mit `return`

```
def summiere(x1, x2):
    summe = x1 + x2
    return summe

res = summiere(3, 5)

print(res)
```



Funktionen: Mit `return`

```
def summiere(x1, x2):
    summe = x1 + x2
    return summe

res = summiere(3, 5)

print(res)
```

Wert (8) an das Hauptprogramm zurückgeben...

Funktionen: Mit `return`

```
def summiere(x1, x2):
    summe = x1 + x2
    return summe

res = summiere(3, 5)

print(res)
```

Wert (8) an das Hauptprogramm zurückgeben...

...*und* Wert in Variable speichern, z.B. `res`

Funktionen: Mit `return`

```
def summiere(x1, x2):
    summe = x1 + x2
    return summe

res = summiere(3, 5)

print(res)
```

Wert (8) an das Hauptprogramm zurückgeben...

...*und* Wert in Variable speichern, z.B. `res`

Weshalb nicht einfach `print` verwenden?

Funktionen

Ein einfaches Beispiel

```
def berechne_rabatt(preis, rabatt_pct):
    rabatt = preis * (rabatt_pct / 100)
    return preis -  rabatt


def berechne_gesamtpreis(preis, rabatt_pct, mwst):
    rabattpreis  = berechne_rabatt(preis, rabatt_pct)
    mwst = rabattpreis * (mwst / 100)
    return  rabattpreis + mwst


endpreis  = berechne_gesamtpreis(100, 15, 7.7)
print("Der Preis nach Rabatt und Mwst ist", endpreis)
```

Funktionen

Ein einfaches Beispiel

```
def berechne_rabatt(preis, rabatt_pct):
    rabatt = preis * (rabatt_pct / 100)
    return preis - rabatt

def berechne_gesamtpreis(preis, rabatt_pct, mwst):
    rabattpreis = berechne_rabatt(preis, rabatt_pct)
    mwst = rabattpreis * (mwst / 100)
    return rabattpreis + mwst


endpreis = berechne_gesamtpreis(100, 15, 7.7)
print("Der Preis nach Rabatt und Mwst ist", endpreis)
```

Wert zurückgeben (und speichern)

Funktionen

Ein einfaches Beispiel

```
def berechne_rabatt(preis, rabatt_pct):
    rabatt = preis * (rabatt_pct / 100)
    return preis - rabatt

def berechne_gesamtpreis(preis, rabatt_pct, mwst):
    rabattpreis = berechne_rabatt(preis, rabatt_pct)
    mwst = rabattpreis * (mwst / 100)
    return rabattpreis + mwst

endpreis = berechne_gesamtpreis(100, 15, 7.7)
print("Der Preis nach Rabatt und Mwst ist", endpreis)
```

Wert zurückgeben (und speichern)

Wert zurückgeben (und speichern)

Funktionen: Beispiel 2

```
def berechne_rechteck_flaeche(laenge, breite):
    flaeche = laenge * breite
    return flaeche

def ist_grosse_flaeche(flaeche,schwellenwert):
    return flaeche > schwellenwert

# Berechne die Fläche
flaeche = berechne_rechteck_flaeche(15, 8)

# Überprüfe, ob die Fläche grösser als 50 ist
test = ist_grosse_flaeche(flaeche, 50)
if test:
    print("Fläche des Rechtecks ist grösser als 50.")
else:
    print("Fläche des Rechtecks ist kleiner 50.")
```

Wo sind Parameter?

Funktionen: Beispiel 2

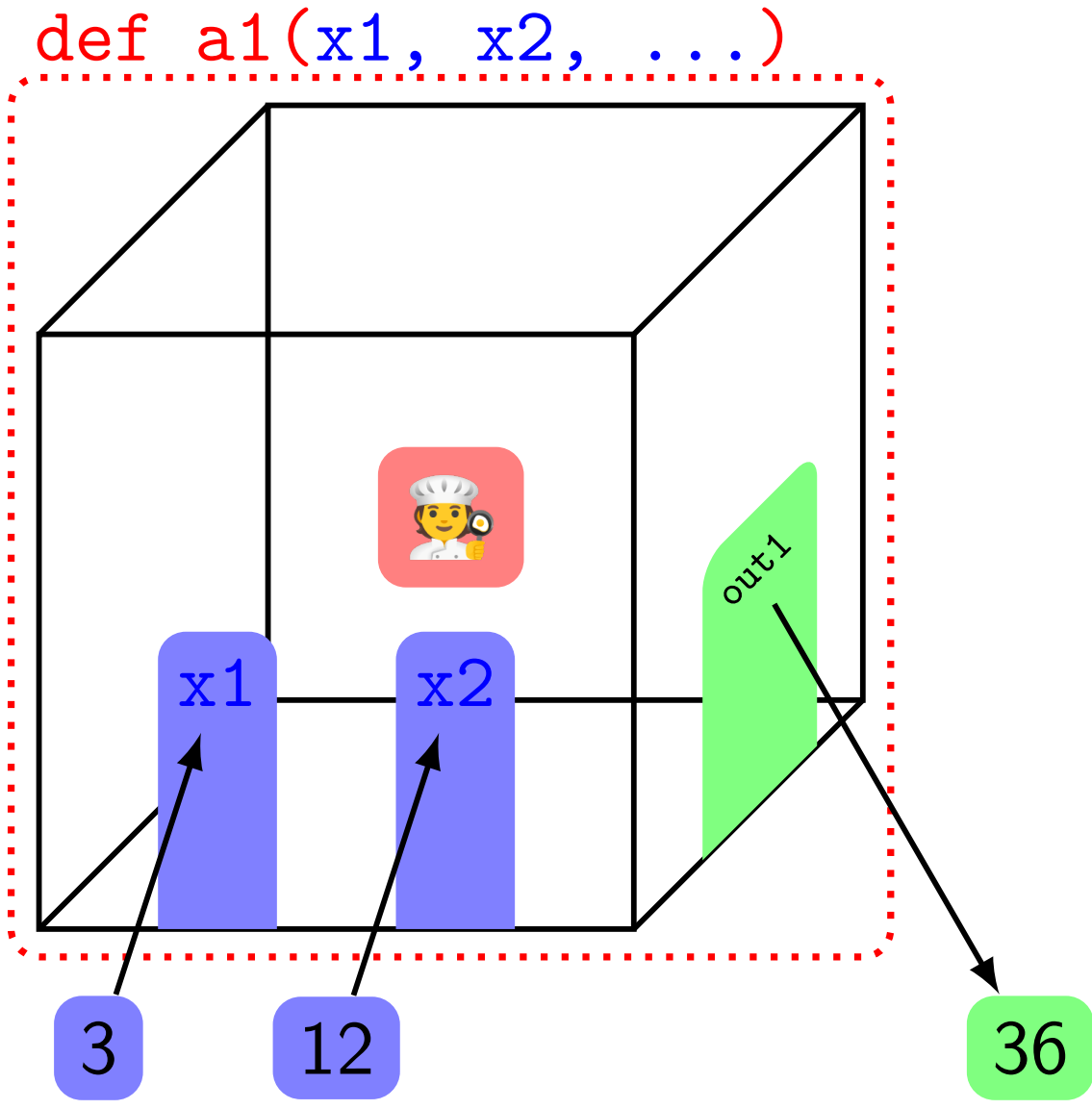
```
def berechne_rechteck_flaeche( laenge, breite ):
    flaeche = laenge * breite
    return flaeche

def ist_grosse_flaeche( flaeche, schwellenwert ):
    return flaeche > schwellenwert

# Berechne die Fläche
flaeche = berechne_rechteck_flaeche( 15, 8 )

# Überprüfe, ob die Fläche grösser als 50 ist
test = ist_grosse_flaeche( flaeche, 50 )
if test:
    print("Fläche des Rechtecks ist grösser als 50.")
else:
    print("Fläche des Rechtecks ist kleiner 50.")
```

Wo sind Return-Werte?

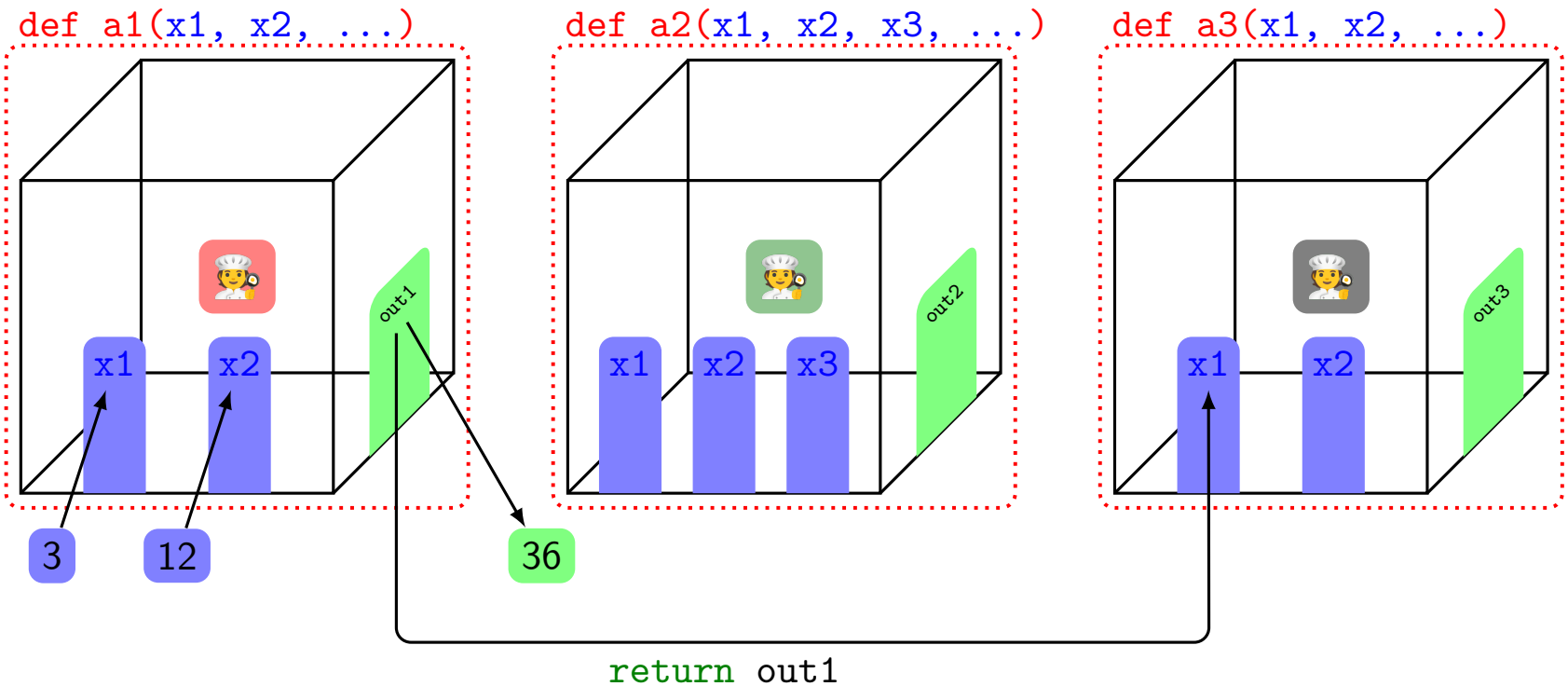
Funktionen: Beispiel 2

```
def berechne_rechteck_flaeche( laenge, breite ):
    flaeche = laenge * breite
    return flaeche

def ist_grosse_flaeche( flaeche, schwellenwert ):
    return flaeche > schwellenwert

# Berechne die Fläche
flaeche = berechne_rechteck_flaeche( 15, 8 )

# Überprüfe, ob die Fläche grösser als 50 ist
test = ist_grosse_flaeche( flaeche, 50 )
if test:
    print("Fläche des Rechtecks ist grösser als 50.")
else:
    print("Fläche des Rechtecks ist kleiner 50.")
```

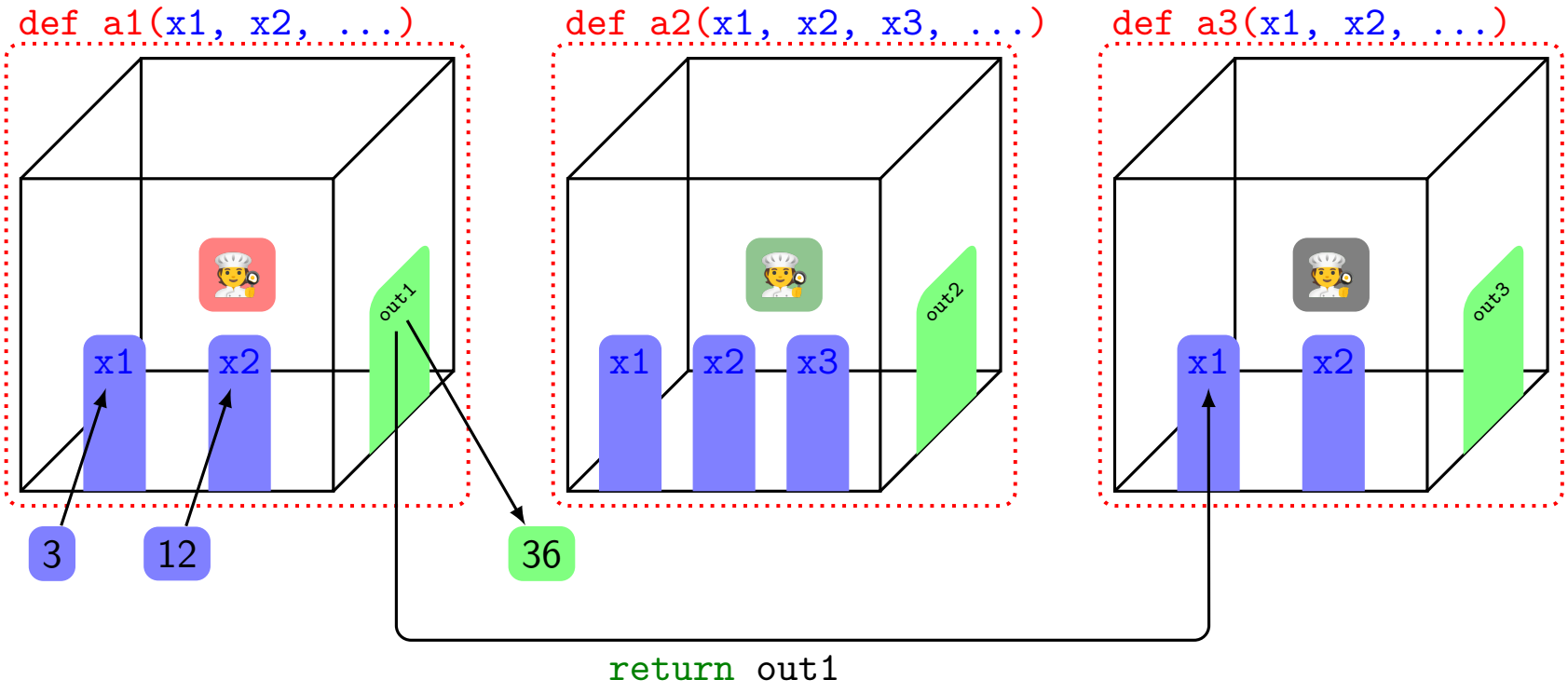
Modularität durch Funktionen



Modularität durch Funktionen

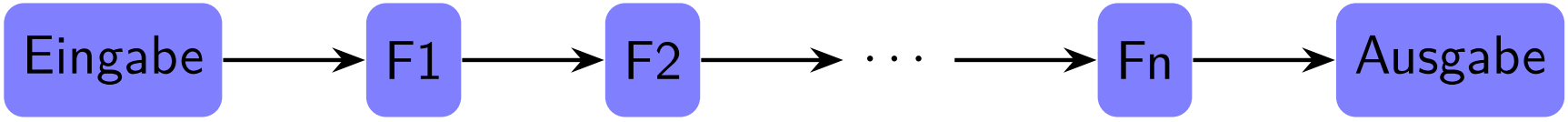


Modularität durch Funktionen



Auch dieses Bild wurde mit einer Funktion erstellt...

Modularität durch Funktionen



Analogien:

- Uhren-Fabrik
- Michelin-Küche
- ...alle komplexen Prozesse, die man in Unter-Prozesse aufbrechen muss!

Anwendungen: Überall! Daten-Analyse, AI, Business Development etc.

Auftrag

Programmier-Skript, Kapitel 8

- Aufgaben 8.1 - 8.2
  1. Zuerst in VS Code schreiben
  2. Danach in Moodle testen: „Kapitel 6 (und 3): return“
- Für Schnelle:
  - Aufgaben 8.3 - 8.4
  - Aufgaben 6.29, 6.30

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)
- 👨‍🍳 ...die wie ein **Koch** funktionieren (**Modularität**):

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)
- 👨‍🍳 ...die wie ein **Koch** funktionieren (**Modularität**):
  - Ein Koch schneidet alle Gemüse...

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)
- 👨‍🍳 ...die wie ein **Koch** funktionieren (**Modularität**):
  - Ein Koch schneidet alle Gemüse...
  - ..., ein Koch grilliert die Gemüse...

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)
- 👨‍🍳 ...die wie ein **Koch** funktionieren (**Modularität**):
  - Ein Koch schneidet alle Gemüse...
  - ..., ein Koch grilliert die Gemüse...
  - ..., ein Koch bereitet die Teller schön zu...

Fazit

- 👍 Vorteil von `return`-Werten: man kann nun das Resultat einer Funktion in einer Variable speichern und **weiterverwenden**
- 👍 Vorteil von Funktionen: Code wird **modular**
- Funktionen sind Definitionen...
- 📋 ...die wie ein **Kochrezept** funktionieren:
  - 🍞 🧀 🍗 Gewisse Inputs, bzw. **Parameter** können akzeptiert werden (müssen aber nicht)
  - 🍔 Gewisse Outputs, bzw. **Return-Werte** können zurückgeben werden (müssen aber nicht)
- 👨‍🍳 ...die wie ein **Koch** funktionieren (**Modularität**):
  - Ein Koch schneidet alle Gemüse...
  - ..., ein Koch grilliert die Gemüse...
  - ..., ein Koch bereitet die Teller schön zu...
  - etc. (jeder Koch ist eine `def`)