

programmierung und datenbanken

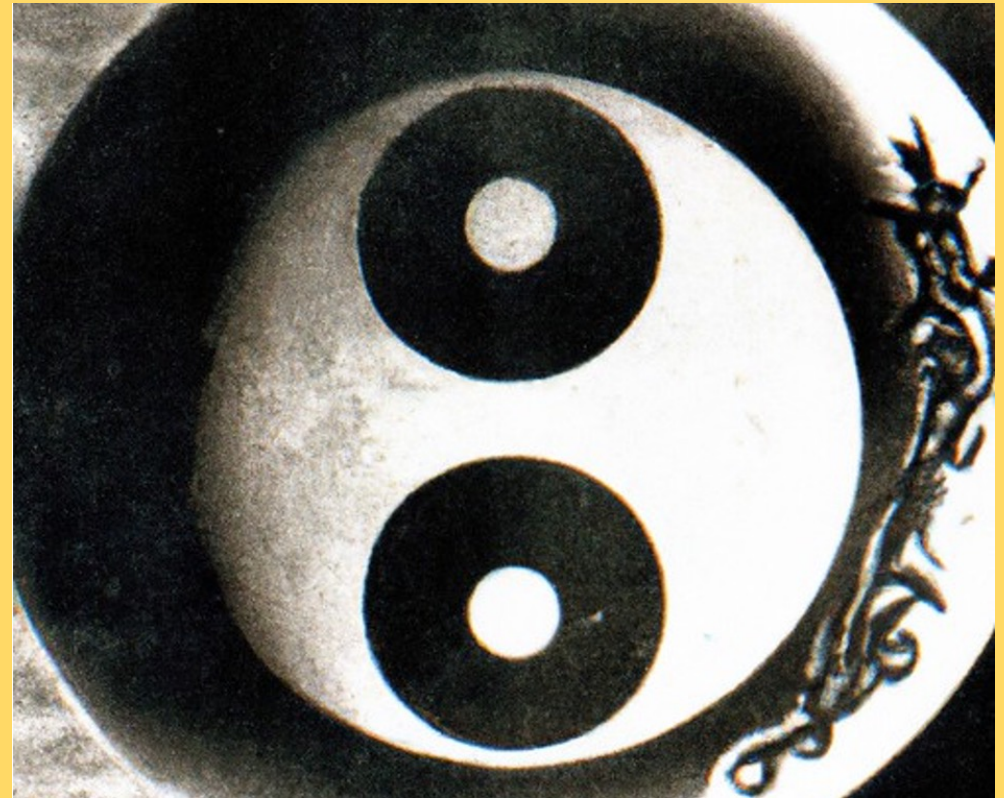
Joern Ploennigs

Funktionen

MIDJOURNEY: FUNCTIONAL LINES, REF. C.E.B. REAS

WIEDERHOLUNG: HÖRSAALFRAGE

Wofür verwendet man if, elif, else?



DALL·E 2: Yin and Yang in a ceramic dish

WIEDERHOLUNG: IF, IFEL, ELSE

if, elif, else sind Befehle zum Verzweigen von Programmen **in** Python

```
if Bedingung1:
    print("Wird ausgeführt wenn Bedingung1 wahr ist")
elif Bedingung2:
    print("Wird ausgeführt wenn Bedingung1 falsch ist und Bedingung2 wahr ist")
else:
    print("Wird ausgeführt wenn Bedingung1 falsch ist und Bedingung2 falsch ist")
```

WIEDERHOLUNG: Hörsaalfrage

Welche Loop Typen gibt es?



Midjourney: Loop

WIEDERHOLUNG: SCHLEIFEN

Schleifen sind ein wichtiger Ansatz in der Programmierung, um wiederholende oder inkrementelle Lösungen zu realisieren.

In Python sind verfügbar:

- `for` (Entspricht For-Each-Loop)
- `while` (Entspricht While-Loop)

Wichtige Eigenschaften:

- Funktionieren genau wie alle anderen Steueranweisungen durch Einrückungen nach einer einzeiligen Definition
- Anders als in Funktionen sind Variablen aus Schleifen auch außerhalb der Schleife verfügbar!
- Die anderen Loop-Varianten sind zwar nicht explizit vorhanden, aber funktionell nachbildbar

WIEDERHOLUNG: WHILE-LOOPS

While-Loop

Im While-Loop kann die Bedingung schon am Anfang falsch sein. Die Schleife muss also nicht ausgeführt werden.

```
Bedingung = True/False
while Bedingung:
    # Statement
    Bedingung = False
```

Do-While-Loop

Im Do-While-Loop wird die Bedingung erst am Ende geprüft. Die Schleife wird also immer mindestens einmal durchlaufen.

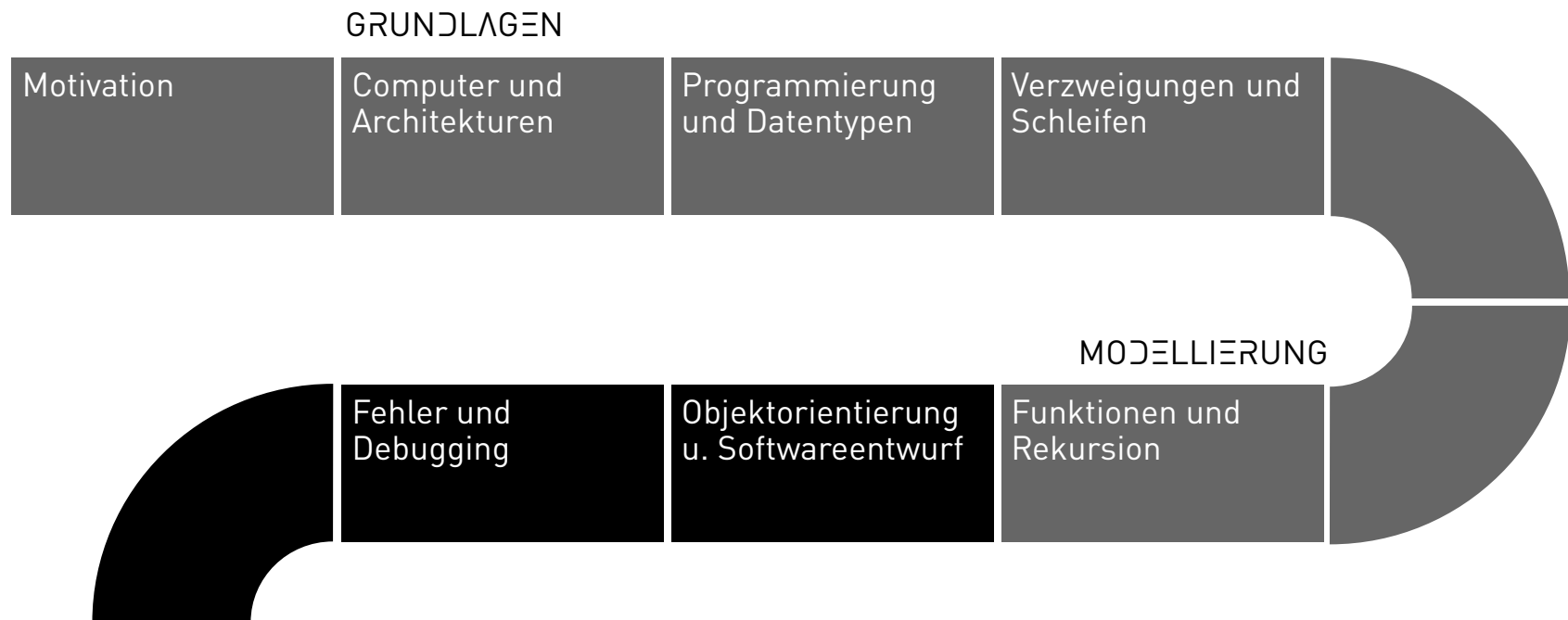
```
Bedingung = True
while Bedingung:
    # Statement
    Bedingung = False
```

Repeat-Until-Loop

Beim Repeat-Until-Loop wird die Bedingung auch erst am Ende geprüft. Die Schleife wird wiederholt bis die Bedingung wahr wird.

```
Bedingung = False
while not Bedingung:
    # Statement
    Bedingung = True
```

ABLAUF



FUNKTIONEN – MATHEMATISCH

Funktionen setzen in Programmiersprachen das mathematische Konzept der Funktion um.

Sie stellen eine Abbildung von einer Eingabemenge auf eine Ausgabemenge dar.

Abbildung: Eine Funktion f ordnet jedem Element x einer Definitionsmenge D ein Element y einer Zielmenge Z zu.

$$f : D \rightarrow Z, \quad x \mapsto y$$

FUNKTIONEN – PROGRAMMIERUNG



Definition: Funktion

Wiederverwendbarer Programmcode, der eine bestimmte Aufgabe ausführt.

- Nehmen ein Tupel an Eingabewerten (*Argumente*)
- Führen eine festgelegte Folge von Ausdrücken und Zuweisungen aus
- Geben ein Tupel an Ausgabewerten zurück (*Rückgabewerte*)
- Werden nur ausgeführt, wenn sie in einem Ausdruck **aufgerufen** werden

FUNKTIONEN IN PYTHON

```
# Codeblock der Funktionsdefinition  
def funktionsname(arg1, arg2): # <- funktionsname ist wählbar, arg1 und arg2  
sind Argumente (Parameter)  
    statement1 # <- Codeblock beginnt hier und ist eingerückt  
    statement2 # <- noch Teil des eingerückten Codeblocks
```

- Funktionsdefinitionen beginnen mit **def**
- Es folgen Name, Argumente und ein **:**
- Argumente gelten im gesamten Funktionskörper
- Beim Aufruf werden die Argumente mit Eingabedaten belegt

FUNKTIONEN MIT AUSGABE

```
def funktionsname (arg1):  
    statement1  
    return ausgabewert
```

- `return` beendet die Ausführung und liefert Ausgabewerte zurück
- „Zurückgeben“ heißt: Der Rückgabewert ersetzt den Funktionsaufruf im ursprünglichen Ausdruck
- Ist kein `return` definiert, gibt die Funktion `None` zurück

BEISPIEL – VERDOPPLUNG EINES WERTS

Multiplikation

```
x = arg1 * 2    # 4
x = arg1 * 4    # 16
x = arg1 * 8    # 128
```

Bitshift (schneller)

```
def mal2(arg1):
    return arg1 << 1

x = arg1 << 1    # $2^1$
x = arg1 << 2    # $2^2$
x = arg1 << 3    # $2^3$
```

BEISPIEL – EUKLIDISCHE DISTANZ

```
from math import sqrt

def distance(a1, a2, b1, b2):
    return sqrt((a1 - b1)**2 + (a2 - b2)**2)

a1, a2 = 2, 3
b1, b2 = 6, 6
x = distance(a1, a2, b1, b2) # 5.0
```

STANDARDWERTE BEI PARAMETERN

```
def funktionsname(arg1, arg2="default"):  
    return statement1
```

- Argumente können Standardwerte besitzen
- Beim Aufruf optional; übergebene Werte überschreiben den Standard

BEISPIEL – MASSEINHEIT AN ZAHL ANHÄNGEN

```
def measurement(number, unit="meters"):
    return str(number) + " " + unit

x = measurement(12)          # "12 meters"
x = measurement(5.5, "kg")  # "5.5 kg"
```

ÜBERGABE VON ARGUMENTWERTEN (KONZEPTE)

Pass-by-value

- Nur **Werte** werden kopiert
- Übergabe-Variablen sind in der Funktion nicht zugreifbar
- „Sicherer“, da keine unerwartete Neubindung

Pass-by-reference

- Referenz auf die **gleichen Daten**
- Werte in der Funktion veränderbar
- Spart Zeit und Speicher

ÜBERGABE VON ARGUMENTWERTEN IN PYTHON

- Mischvariante, oft „**Pass-by-assignment**“ genannt
- **Mutable** Datentypen: verhalten sich wie *pass-by-reference*
- **Immutable** Datentypen: verhalten sich wie *pass-by-value*
- Wird eine *mutable* Variable **neu gebunden**, wirkt das **nicht** außerhalb

BUILT-IN FUNKTIONEN

Python hat eine umfangreiche Liste eingebauter Funktionen.

Funktion	Zweck
<code>print()</code> , <code>input()</code>	Ausgabe, Eingabe
<code>id()</code> , <code>type()</code>	Variablen ID, Datentyp
<code>int()</code> , <code>str()</code> , <code>float()</code>	Datentyp-Konvertierung
<code>list()</code> , <code>tuple()</code> , <code>set()</code> , <code>dict()</code>	Komplexe Datentypen
<code>len()</code>	Länge eines komplexen Datentyps
<code>abs()</code> , <code>max()</code> , <code>min()</code>	Mathematische Grundfunktionen
<code>exit()</code>	Programm beenden
<code>sorted()</code>	Sortieren
...	

WOZU WERDEN FUNKTIONEN GENUTZT?

- Übersichtlichkeit, Modularität, Wiederverwendbarkeit
„Write once, use anywhere“
- Moderner Stil: Programme so weit wie möglich in **grundlegende Funktionen** aufteilen

fragen?

