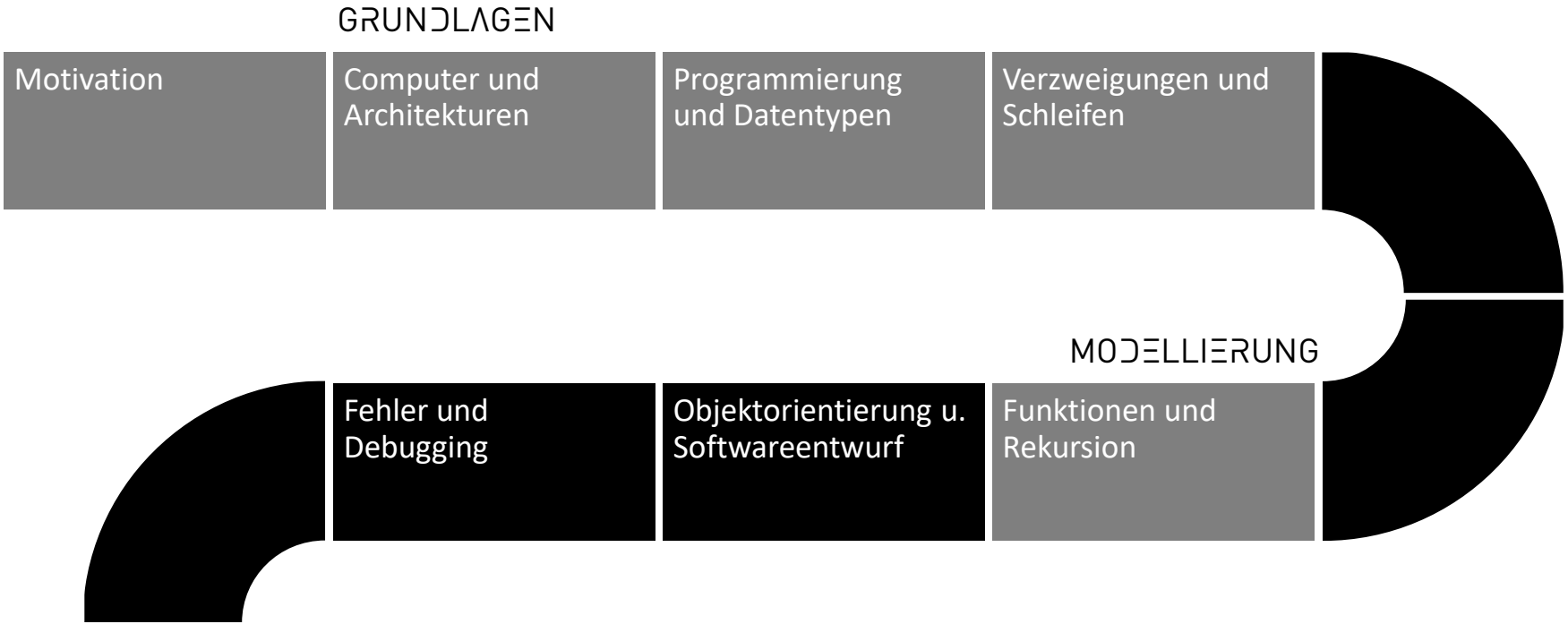
Einführung in Programmierung und Datenbanken

Joern Ploennigs

Programmierung und Datenbanken



Funktionen

Midjourney: Vaulted Ceiling of Fractal Function

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 eine Definitionsmenge D, ein Element y und eine Zielmenge Z zu.

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

Funktionen - Programmierung

- Wiederverwendbarer Programmcode der eine bestimmte Aufgabe ausführt

- Funktionen (auf unserem derzeitigen Wissensstand):
  - Nehmen ein Tupel an Eingabewerten (Argumente)
  - Führen eine festgelegte Folge von Ausdrücken und Zuweisungen aus
  - Geben ein Tupel an Ausgabewerten zurück (Rückgabewerte)

- Funktionen werden nur ausgeführt wenn sie in einem Ausdruck aufgerufen werden.

Funktionen In Python

- Funktionsdefinitionen beginnen mit def
- Es folgt ein Name (ähnlich wie eine Variable), ein Tupel mit Variablen und ein „:“
- Diese Variablen (“Argumente“) sind für den ganzen Rest der Funktion gültig
- Argumente werden beim Aufruf der Funktion mit den Eingabedaten belegt

```
def funktionsname(arg1, arg2):
    Statement1
    Statement2
```

Einrücken →

Codeblock der Funktionsdefinition

Funktionen Ausgabe

- In Funktionen kann die Anweisung return genutzt werden.
- Diese beendet die Ausführung der Funktion und gibt Ausgabewerte zurück.

```
def funktionsname(arg1):
    Statement1
    return Ausgabewert
```

- „Zurückgeben“ heißt hier: Setzt den Wert von Statement2 in den ursprünglichen Aufruf ein, so als wäre der Funktionsaufruf eine Variable (Der Wert kann dann weiterverwendet werden).
- Ist kein return definiert, gibt die Funktion none zurück

Funktionen – Beispiel: Verdopplung eines Wertes

Definieren der Funktion:

```
def mal2(arg1):
    return arg1 * 2
```

Die Funktion aufzurufen funktioniert folgendermaßen:

```
x = mal2(2)     # x ist nun 4

x = mal2(x)     # x ist nun 8
```

Funktionen – Beispiel: Verdopplung eines Wertes

Definieren der Funktion:

```
def mal2(arg1):
    return arg1 << 1   # das geht auch
```

Die Funktion aufzurufen funktioniert folgendermaßen:

```
x = mal2(2)     # x ist nun 4

x = mal2(x)     # x ist nun 8
```

Funktionen – Beispiel: Euklidische Distanz

Definieren der Funktion (mit Nutzung der Quadratwurzel durch sqrt Funktion)

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

Punkte definieren und Funktion aufrufen:

```
a1, a2 = 2, 3
b1, b2 = 6, 6
x = distance(a1, a2, b1, b2)      # x ist nun 5
```

Funktionen - Default Values

- Beim Definieren einer Funktion können den Argumenten Standardwerte gegeben werden.
- Dadurch werden diese beim Aufruf der Funktion optional. Werden sie genutzt wird der Standardwert überschrieben.

```
def funktionsname(arg1, arg2 = “default“):
    return Statement1
```

Funktionen – Beispiel: Maßeinheitt zu Zahl hinzufügen

Definieren der Funktion (mit der str Funktion, die eine Zahl in einen String überführt)

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

Funktion aufrufen:

```
x = measurement(12)              # x ist nun “12 meters“

x = measurement(5.5, “kg“)    # x ist nun “5.5 kg“
```

Funktionen - Übergeben von Argumentwerten

Pass-by-reference

- Geben wir einer Funktion Variablen als Argumente, sind diese in der Funktion voll zugreifbar (veränderbar).
- Benötigt weniger Zeit und Speicher.

Pass-by-value

- Beim Aufrufen der Funktion werden nur die Werte der Variablen kopiert.
- Die eingegebenen Variablen sind in der Funktion nicht zugreifbar.
- Ist „sicherer“, da Variablen nicht unerwartet neu belegt werden.

Funktionen - Übergeben von Argumentwerten in Python

- Nutzt „Mischvariante“, meistens bezeichnet als *Pass-by-assignment*

- Variablen mit *mutable* Datentyp: Pass-by-reference
- Variablen mit *immutable* Datentyp: Pass-by-value

- Wird eine Variable mit mutablen Datentyp jedoch in der Funktion ganz neu belegt, wird dies außerhalb der Funktion *nicht* übernommen.

Funktionen – Built-in Python

- Python hat eine Liste an eingebauten Funktionen

`print(), input()`	Ausgabe, Eingabe
`id(), type()`	Variablen ID, Variablen Datentyp
`int(), str(), float()`	Datentyp Konvertierung
`list(), tuple(), set(), dict()`	Komplexe Datentypen
`len()`	Länge eines komplexen Datentyps
`abs(), max(), min()`	Mathematische Grundfunktionen
`exit()`	Programm Beenden
`sorted()`	Sortieren
…	

Funktionen - Wozu werden sie genutzt?

- Übersichtlichkeit, Modularität, Wiederverwendbarkeit

„Write once, use anywhere“

- Moderner Programmierstil: Programme so weit wie möglich in grundlegende Funktionen aufteilen

LITERATURHINWEISE

- Umfassende Erklärung: https://docs.python.org/3/reference/datamodel.html
- Python Standardfunktionen: https://docs.python.org/3/library/functions.html

HÖRSAALFRAGE

FRAGEN?

DALL·E 2: A psychedelic DJ with a question mark for a head