Konzepte des Prozeduralen Programmierens

Prof. Dr. Sascha Frohwerk

2023-09-11

Contents

Vorwort

Dies ist das Skript mit Beispielen und kurzen Erklärungen zur Vorlesung "Konzepte des prozeduralen Programmierens" an der FOM in Leipzig im Wintersemester 2023 / 2024. Weil das Skript aktuell aus einem anderen Format übertragen und dabei auch überarbeitet wird, liegt es noch nicht vollständig vor. Es wird sich während des Semsters weiterentwickeln und ist dann rechtzeitig vor der Klausur fertig sein.

Chapter 1

Einleitung

-> siehe Folien

Chapter 2

Eingabe, Ausgabe und einfache Operatoren

2.1 Hallo EVA

Bei einfachen Programmen sollte man sich (wenn möglich) an das EVA-Prinzip halten. Das erhöht die Lesbarkeit und macht die weitere Bearbeitung des Programms einfacher.

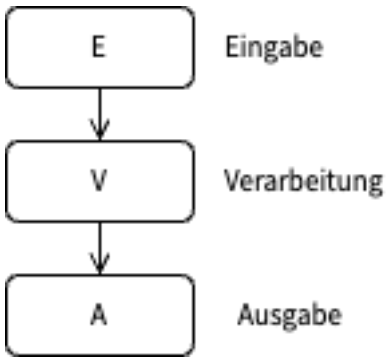

Figure 2.1: Das EVA-Prinzip

Der folgende Code zeigt ein einfaches Programm, das den Benutzer um die Eingabe einer ganzen Zahl bittet, diese mit zwei multipliziert und wieder ausgibt. Das Programm folgt dem EVA-Prinzip: Eingabe - Verarbeitung - Ausgabe. Dies hilft Code zu strukturieren und wir wollen versuchen, diesem Prinzip zu folgen, wenn es möglich ist.

```
# Eingabe
```

```
eingabe = input("Bitte eine Zahl eingeben:")

# Verarbeitung
zahl = int(eingabe)
ausgabe = 2*zahl

# Ausgabe
print(ausgabe)
```

Anmerkungen dazu:

- Anweisungen werden von oben nach unten abgearbeitet
- "=" weist einer Variablen einen Wert zu
- In Python müssen diese Variablen vorher nicht definiert werden
- input() liest Werte ein
- print() gibt Werte aus
- int() wandelt einen String in einen Integer um
- Zeilen mit einer Raute am Anfang sind Kommentare. Sie werden nicht übersetzt.

Die Namen von Variablen oder Funktionen nennen wir Bezeichner, hier z.B. "eingabe". Diese dürfen in Python mit einem Buchstaben oder einen Unterstrich beginnen (letzteres ist aber nicht zu empfehlen), aber nicht mit einer Zahl oder einem Sonderzeichen. Sie dürfen ausserdem nicht den Namen eines Schlüsselwortes haben. Schlüsselworte sind Worte, die in Python eine besondere Bedeutung haben, z.B. dürfen Sie Ihre Variable nicht "while" nennen, weil damit eine Schleife eingeleitet wird.

Weitere Tips zum Umgang mit Code

Anweisungen in einer Zeile zusammenfassen:

```
a = 3; b = 6;
print(a+b)
```

```
## 9
```

Lange Zeilen umbrechen:

```
a = \
5
print(a)
```

```
## 5
```

Ein- und Ausgabe von Zeichenketten (Strings):

```
print("Dein Name:")
name = input()
print("Hallo "+name)
```

Seit Python 3.6 gibt eine weitere Möglichkeit Strings und andere Variablen zu verknüpfen: f-Strings

```
name = "Klaus"
print(f"Hallo {name}")
```

```
## Hallo Klaus
```

2.2 Einfache Datentypen

Man unterscheidet zwischen einfachen und strukturierten Datentypen. Einfache Datentypen können eine einzige Information, zum Beispiel eine Zahl, speichern. Strukturierte Datentypen können mehrere Zusammenhänge Daten speichern. Hierbei kann es sich zum Beispiel um Listen oder Tabellen handeln. Die folgende Abbildung gibt einen Überblick über die in Python vorhandenen Datentypen.

Datentypen allgemein

In allen Programmiersprachen und Datenbanksystemen gibt es Datentypen. Sie definieren, welche Art von Daten eine Variable oder eine Spalte in einer Tabelle aufnehmen kann. Allgemein gibt es folgende grundlegende Datentypen:

Table 2.1: Datentypen allgemein

Typ	Python	C	MySQL
nichts	None	null, void	null
ganze Zahlen	int	int	int
Fließkommazahlen	float	float, double	float

Typ	Python	C	MySQL
komplexe Zahlen	complex	float complex	-
Buchstaben	bytearray	char	char
Zeichenketten	string	-	varchar
Wahrheitswerte	bool	-	boolean

In C gibt es nativ keinen Datentyp für Wahrheitswerte. Dies wird über die Integer-Werte 0 (falsch) und 1 (wahr) ausgedrückt. Seit C99 gibt es _Bool und bool. Es gibt von jedem Grundtyp noch mehrere Varianten in Abhängigkeit von der Größe (Länge) des Wertes und ob negative Werte erlaubt sind. Z.B: (in C)

```
long long a;
long double b;
unsigned int c;
```

Datentypen in Python

Der Datentyp wird in Python dynamisch vom Interpreter festgelegt und nicht vom Entwickler. Dennoch haben alle Variablen natürlich einen Datentyp. Dieser kann mit der Funktion type() bestimmt werden:

```
a = 4
print(type(a))
```

```
## <class 'int'>
```

Wir sehen:

- Der Variablen wird automatisch der Typ zugewiesen, den der Anfangswert hat.
- In Python sind alles Objekte, auch normale Datentypen. In der Praxis merken wir davon meist wenig.

```
a = 4
print(type(a))
```

```
## <class 'int'>
```

```
a = a / 3
print(a)
```

```
## 1.3333333333333333
```

```
print(type(a))
```

```
## <class 'float'>
```

Wir sehen: Der Datentyp wird bei Bedarf dynamisch verändert.

Die Menge verfügbarer Datentypen kann mit eingebundenen Bibliotheken erweitert werden.

2.3 Operatoren zur Berechnung

Die Operatoren zur Berechnung entsprechen im wesentlichen denen aus der Mathematik. Zusätzlich gibt es ganzzahliges Teilen ("//") und Modulo ("%"). Potenzen werden mit "**" ausgedrückt, also nicht "^", wie man es etwas aus Excel kennt.

```
a = 5
b = 2
print(a+b)
```

```
## 7
```

```
print(a-b)
```

```
## 3
```

```
print(a/b)
```

```
## 2.5
```

```
print(a//b)
```

```
## 2
```

```
print(a%b)
```

```
## 1
```

```
print(a**b)
```

```
## 25
```

Für Zuweisungen und Operatoren gibt es eine verkürzte Schreibweise, z.B. a +=5 statt a = a + 5.

```
a = 5
a += 1
print(a)
```

```
## 6
```

2.4 Vergleichsoperatoren

Auch die Vergleichsoperatoren lehnen sich an die Mathematik an.

Table 2.2: Vergleichsoperatoren

Syntax	Bedeutung
==	gleich
!=	ungleich
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich

Beispiele:

```
a = 10
print(a < 20 and a > 7)
```

```
## True
```

oder auch

```
a = 7 < 10 < 20
print(a)
```

```
## True
```

Die Zahl 0 wird als False gewertet.

```
a = bool(5-5)
print(a)
```

```
## False
```

Chapter 3

Kontrollstrukturen

Chapter 4

Strukturierte Datentypen

Chapter 5

Modularisierung

Chapter 6

Weitere Themen

Chapter 7

Beispiele angewandter Programmierung