Einführung in die Programmierung

Ronja Düffel
WS2017/18

05. Oktober 2017

Programmieren (vereinfacht)

1. Problem beschreiben und analysieren
2. Enwicklung und Beschreibung einer Lösung
3. Übertragung/Umsetzung in eine Programmiersprache
4. Test des Programms

Programmiersprachen

Maschinenprogramme

- können direkt vom Computer verstanden und ausgeführt werden.
- bestehen aus Bit-Folgen (0-en und 1-en),
- für Menschen nahezu unverständlich

Höhere Programmiersprachen

- für Menschen besser zu lesen und zu verstehen
- *Quelltext* = Programm in höherer Programmiersprachen
- für Computer unverständlich

  $\Rightarrow$ Quelltext muss in Maschinenprogramm übersetzt werden!

Formale und natürliche Sprachen

formale Sprache	**natürliche Sprache**
$3 + 5 = 8$ oder H_2O	Englisch, Deutsch, ...
von Menschen entworfen	haben sich natürlich entwickelt

Syntax, Semantik

Eindeutigkeit	Mehrdeutigkeit
keine Redundanz, präzise	Redundanz

Informationsdichte in formaler Sprache viel höher als in natürlicher Sprache

Python

- Höhere Programmiersprache
- Als Lehrsprache entwickelt

  → Übersichtlichkeit

  → leichte Erlernbarkeit
- benutzt zur Übersetzung einen Interpreter

IDLE

- Entwicklungsumgebung für Python
- unterstützt bei der Programmierung
- Interaktive Python-Shell mit
  - Syntax-Highlighting
  - Autovervollständigung
  - Anzeige der Befehlssyntax
- Editor zum Programme schreiben

Datentypen

Wahrheitswerte `Bool`

- `True` `False`
- Operatoren: `and`, `or` und `not`

Zahlen

- Ganzzahlen `int` (fehlender Dezimalpunkt)
- Gleitkommazahlen `float` (Dezimalpunkt)
- Operatoren: `+`, `-`, `*`, `/`, `//`, `%`, `**`

Zeichenketten `string`

- in Anführungszeichen

```
'Und nun zu etwas ganz anderem...'
''Setz dich, nimm dir'n Keks...''
'''Er sagte:''Geht's jetzt los?'' '''
```

Built-in-Funktionen (`int`, `float`)

- `abs(x)`: gibt den Betrag $|x|$ der Zahl x zurück
- `max(a,b)`: gibt das Maximum von a und b zurück
- `min(a,b)`: gibt das Minimum von a und b zurück
- `round(x,n)`: gibt den gerundeten Wert von x zurück (n-Nachkommastellen)

Built-in-Funktionen (`string`)

- `<string>.isalpha()`: prüft, ob String nur aus Buchstaben besteht
- `<string>.isdigit()`: prüft, ob String nur aus Ziffern besteht
- `ord(c)`: gibt die Nummer des Zeichens `c` zurück
- `chr(<int>)`: gibt das Zeichen zur Ganzzahl `i` zurück
- `+`: fügt zwei Strings zusammen (Operator)

nützliche Funktionen

- `int(x)`: wandelt `x` in einen ganzzahligen Wert um
- `float(x)`: wandelt `x` in einen Gleitkommawert um
- `string(x)`: wandelt `x` in einen String um
- `input()`: Benutzereingabe lesen
- `print()`: Bildschirmausgabe

Variablen

- Möglichkeit Informationen zu speichern
- bezeichnet einen bestimmten Speicherbereich
- hat immer:
  - einen Namen
  - einen Typ
  - einen Wert
- in Python durch Zuweisung `<Name> = <Ausdruck>` angelegt

! Vergibt man denselben Namen zweimal, so wird der Wert der Variablen überschrieben !

Variablennamen

- von Programmierer* in vergebene Namen müssen mit Buchstaben (a,. . . ,z,A,. . . , Z) oder Unterstrich (_) beginnen.
- können beliebig lang sein und ab 2.Zeichen Ziffern(0,. . . , 9) enthalten.
- Groß- und Kleinschreibung ist immer relevant!
- Umlaute (ä,ö,ü,. . .) sollten nicht verwendet werden
- Sonderzeichen(!,&, . . .) sind nicht erlaubt
- Schlüsselwörter sind als Variablennamen verboten (z.B. and, or, if, in . . .)

Namenskonventionen

- **Variable:** nomen_mit_unterstrich
  Bsp: current_line, radius_circle, ...

- **Konstanten:** NOMEN_IN_GROSSBUCHSTABEN
  Bsp: MAX_LENGTH, MIN_RADIUS,...

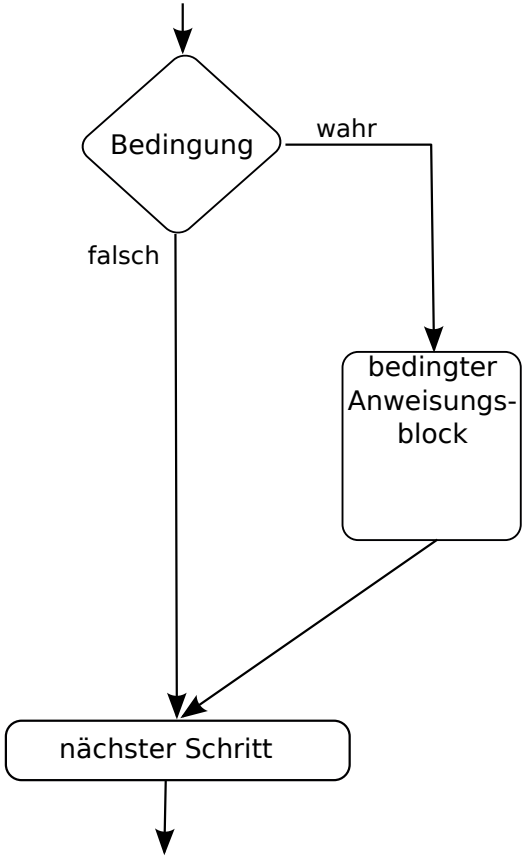
- **Tipps**:
  - gleich angewöhnen Englische Namen zu wählen
  - möglichst beschreibende Namen wählen:
    z.B. `current_line` statt `cl`
  - Singular für einzelene Objekte, Plural für Sammlungen
    (`student_name` vs `student_names`)

Kontrollstrukturen

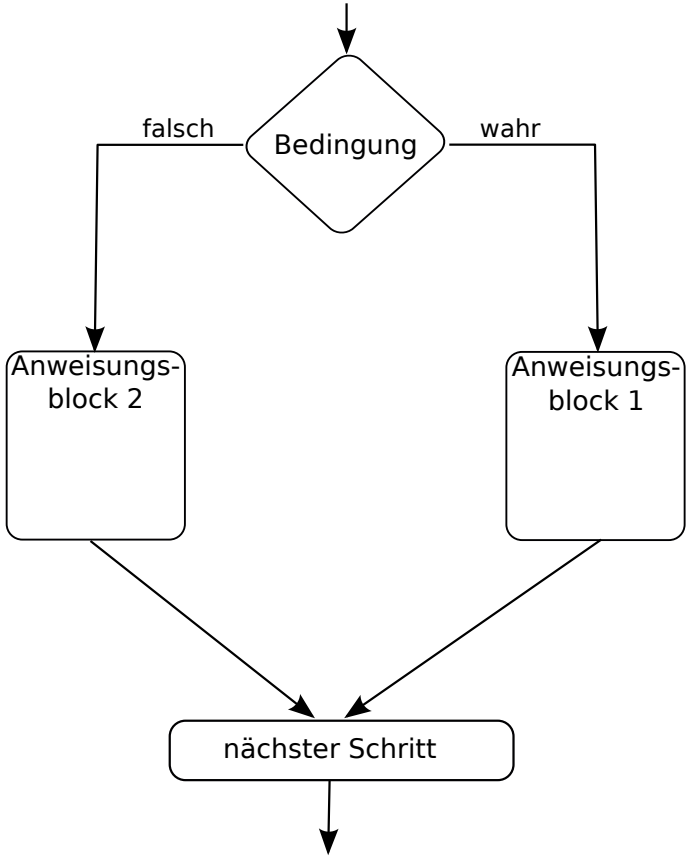
- steuern den Programmablauf

Verzweigung

if-Anweisung

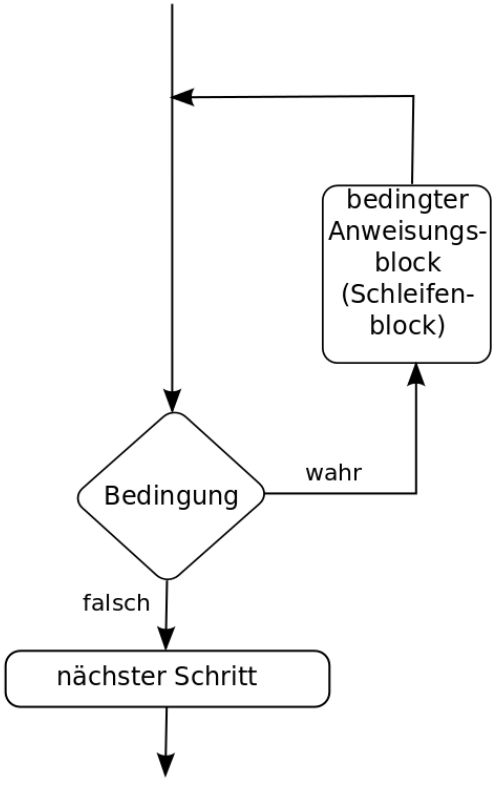


if...else-Anweisung

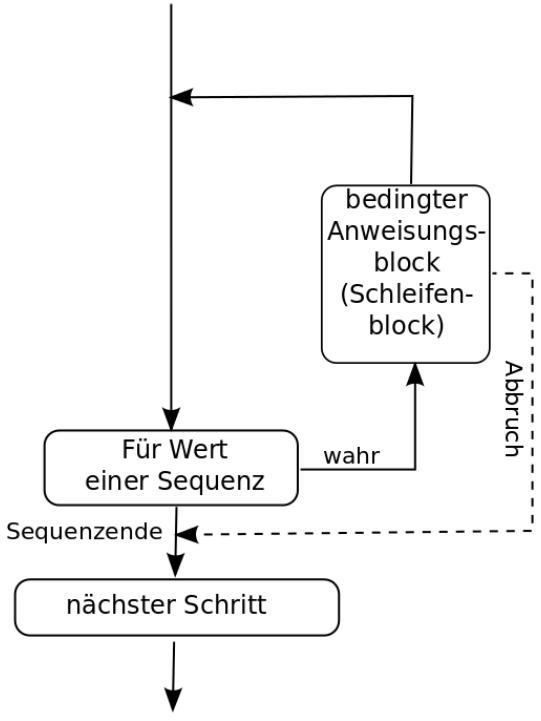


Schleifen

while-Schleife



for-Schleife



`range()`-Funktion

```
range(start, stop[, step])
```

- erzeugt eine Sequenz von Zahlen von `start` bis `stop`-1 (!!!)
- wird `start` nicht angegeben, beginnt die Sequenz bei 0
- wird keine Schrittweite (`step`) angegeben, wird immer um 1 erhöht

Schleifen-Kontrollanweisungen

- `break`: beendet Schleife sofort; Programmfluss springt zur ersten Anweisung nach dem Schleifenrumpf
- `continue`: bricht aktuellen Schleifendurchlauf ab; Programmfluss springt zum Schleifenkopf
- `pass`: wenn Python eine Anweisung verlangt, aber nichts gemacht werden soll