

Informatik I: Einführung in die Programmierung

6. Python-Programme schreiben, kommentieren, starten und entwickeln

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Bernhard Nebel

28. Oktober 2016



Programme

- Programme = konkretisierte Algorithmen?
- Ja, aber nicht immer! Oft eingebettet in Programme.
- Folge von Anweisungen und Ausdrücken, die einen bestimmten Zweck erfüllen sollen.
- Interaktion mit der Umwelt (Benutzer, Sensoren, Dateien)
- Unter Umständen nicht terminierend (OS, Sensorknoten, ...)
- Auf jeden Fall länger als 4 Zeilen!



Programme schreiben

- Zum Schreiben von Programmen benutzt man einen **Texteditor** (**kein** Textverarbeitungssystem wie MS-Word!):
 - *notepad* (Windows)
 - *notepad++* (Windows, Open Source)
 - *vim* (Open Source)
 - *emacs* (Open Source)
 - *gedit* (Open Source)
 - in IDE integrierter Editor (kommt noch)
 - Möglichst mit integriertem Syntaxchecker
- alle bis auf *notepad* haben dies oder unterstützen Plugins für Python

- Umbrechen, wenn Zeilen zu lang.
- Implizite Fortsetzung mit öffnenden Klammern und Einrückung (siehe PEP8):

Lange Zeilen

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)  
  
def long_function_name(  
    var_one, var_two, var_three,  
    var_four):  
    print(var_one)
```

- Explizite Fortsetzung mit *Backslash*:

Explizite Fortsetzung

```
foo = long_var_name1 + long_var_name2 + \  
    long_var_name3
```



- Kommentiere dein Programm!
- Programme werden öfter **gelesen** als geschrieben!
- Auch für ein selbst: Erinnerungen daran, was man sich gedacht hat.
- Nicht das offensichtlich kommentieren, sondern Hintergrundinformationen geben.
- Möglichst englisch kommentieren.
- Der Rest einer Zeile nach # wird als Kommentar interpretiert.

- Blockkommentare: Zeilen, die jeweils mit # beginnen und genauso wie die restlichen Zeilen eingerückt sind beziehen sich auf die folgenden Zeilen.

Block-Kommentare

```
def fib(n):  
    # this is a double recursive function  
    # runtime is exponential in the argument  
    if n == 0:  
  
    ...
```

- Fließtext-Kommentare kommentieren einzelne Zeilen.

Schlechte und gute Kommentare

```
x = x + 1 # Increment x  
  
y = y + 1 # Compensate for border
```


- #-Kommentare sind nur für den Leser.
- Möchte man dem Benutzer Informationen geben, kann man `docstring`-Kommentare nutzen.
- Ist der Ausdruck in einer Funktion oder einem Programm (Modul) ein String, wird dieses der `docstring`, der beim Aufruf der Funktion `help` ausgegeben wird.
- Konvention: Benutze den mit drei `"`-Zeichen eingefassten String, der über mehrere Zeilen gehen kann.

`docstring`

```
def fib(n):  
    """Computes the n-th Fibonacci number.  
    The argument must be a positive integer.  
    """  
    ...
```



- Nachdem man ein Programm eingetippt hat, sollte man es abspeichern.
- Lege ein Verzeichnis in deinem *Home*-Verzeichnis an, und speichere alle deine Programme da.
- Füge dem Dateinamen immer die Dateierweiterung `.py` an, damit man weiß, dass es sich um ein Python-Programm handelt.
- *Windows*: Wähle immer *Alle Dateien* beim Sichern damit nicht `.txt` angehängt wird.



Programme starten

5 Wege ein Programm zu starten



- Starten mit explizitem Aufruf von Python3
- Starten als Skript
- Starten durch Klicken
- Starten durch Import
- Starten in einer IDE

Beispielprogramm: `example.py`

```
print("Hello world")
```



Shell

```
# python3 example.py  
Hello world
```

- Voraussetzungen:
 - Wir sind in dem Ordner, in dem die Datei `example.py` liegt.
 - Die Pfad-Variable (`PATH`) wurde so gesetzt, dass der Python-Interpreter gefunden wird.
- Wird normalerweise bei der Installation geleistet.
- Kann „per Hand“ nachgetragen werden:
 - *Windows*: Systemsteuerung → System und Sicherheit → Erweiterte Systemeinstellungen → Erweitert → Umgebungsvariablen
 - *Unix*: Setzen der `PATH`-Variable im entsprechenden Login-Skript oder in der Shell-Konfigurationsdatei (z.B. `~/.bash_profile`)

Shell

```
# example.py  
Hello world
```

■ Voraussetzungen:

- Wir sind in dem Ordner, in dem die Datei `example.py` liegt.
- *Windows*: `.py` wurde als Standard-Dateierweiterung für Python registriert.
- *Unix*: Die erste Zeile in der Datei `example.py` ist:
`#!/usr/bin/env python3`
und die Datei hat das `x`-Bit (ausführbare Datei) gesetzt.



- Wenn `.py` als Standard-Dateierweiterung für Python registriert ist (geht eigentlich bei allen Plattformen mit Desktop-Oberfläche), kann man die Datei durch Klicken (oder Doppelklicken) starten.
- Leider wird nur kurz das Shell-Fenster geöffnet, mit Ende des Programms verschwindet es wieder.
- *Abhilfe*: Am Ende die Anweisung `input()` in das Programm schreiben.
- *Allerdings*: Bei Fehlern verschwindet das Fenster trotzdem, und man kann keine Parameter beim Aufruf übergeben.
- Eigentlich nur für fertig entwickelte Programme mit GUI geeignet.

- Nachdem wir Python im Ordner aufgerufen haben, in dem `example.py` liegt:

Python-Interpreter

```
>>> import example  
Hello world
```

- *Beachte:* Angabe ohne die Dateierweiterung!
- Funktioniert nur beim ersten Import.

Python-Interpreter

```
>>> import example  
Hello world  
>>> import example  
>>>
```




Programme entwickeln

IDE = Integrated development environment



Einen Editor aufrufen, dann das Programm in der Shell starten, dann wieder den Editor starten, ...

Stattdessen kann man IDEs einsetzen für:

- Projektverwaltung
- Programm editieren
- Ausführen
- Testen und *Debuggen*
- Dokumentation erzeugen
- ...

Gibt es in den verschiedensten Komplexitäts- und Qualitätsabstufungen.



Wohlmöglich benannt nach Eric Idle.

- Ist 100% in Python geschrieben und benutzt die *tkinter* GUI (graphical user interface).
- Läuft auf allen Plattformen.
- Multi-Fenster-Texteditor mit Syntaxkennzeichnung, multipler Zurücknahme, smarterer Einrückung.
- Enthält ein Fenster mit Python-Shell.
- Rudimentäre Debug-Möglichkeiten.
- Beschreibung siehe:
<http://docs.python.org/3/library/idle.html>.



- **File-Menü:** New, Open und Recent File zum Öffnen einer neuen bzw. vorhandenen Programmdatei.
- **File-Menü:** Save und Save as abhängig davon, welches Fenster aktiv. Entweder die Shell-Interaktionen oder die Programmdatei wird gespeichert.
- **Shell-Menü:** Nur im Shell-Fenster aktiv. Hier kann man mit Restart den Interpreter neu starten.
- **Run-Menü:** Ist nur im Editorfenster aktiv. Hier kann man die Syntax überprüfen und das Programm starten, nachdem der Interpreter neu gestartet wurde.



Beispiel: Interaktiver Taschenrechner



- Wir wollen ein Skript schreiben, dass wiederholt



- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und



- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,



- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,



- Wir wollen ein Skript schreiben, dass wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.



- Wir wollen ein Skript schreiben, dass wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.
- Erst einmal nur für + und -



- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.
- Erst einmal nur für + und -
- Dabei nutzen wir die Funktion `input(String)`, die eine Benutzereingabe erwartet und diese als String zurück gibt.

- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.
- Erst einmal nur für + und -
- Dabei nutzen wir die Funktion `input(String)`, die eine Benutzereingabe erwartet und diese als String zurück gibt.

Python-Interpreter

```
>>> input('Deine Eingabe:')  
Deine Eingabe:
```

- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.
- Erst einmal nur für + und -
- Dabei nutzen wir die Funktion `input(String)`, die eine Benutzereingabe erwartet und diese als String zurück gibt.

Python-Interpreter

```
>>> input('Deine Eingabe:')  
Deine Eingabe:
```

- Wir wollen ein Skript schreiben, das wiederholt
 - nach zwei Operanden und
 - einem arithmetischen Operator fragt,
 - dann die Operation ausführt,
 - und das Ergebnis ausgibt.
- Erst einmal nur für + und -
- Dabei nutzen wir die Funktion `input(String)`, die eine Benutzereingabe erwartet und diese als String zurück gibt.

Python-Interpreter

```
>>> input('Dein Eingabe:')  
Deine Eingabe:blau  
blau
```

Python-Programm

```
while (True):
    op1 = float(input("1. Operand: "))
    op2 = float(input("2. Operand: "))
    opa = input("Operator:  ")
    if opa == "+":
        print(op1 + op2)
    elif opa == "-":
        print(op1 - op2)
    else:
        print("Falscher Operator")
```


break und continue

- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).



break und continue

- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).
- Wir würden gerne (kontrolliert) die `while`-Schleife verlassen!



break und continue

- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).
- Wir würden gerne (kontrolliert) die `while`-Schleife verlassen!
- Dafür gibt es `break`:



break und continue



- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).
- Wir würden gerne (kontrolliert) die `while`-Schleife verlassen!
- Dafür gibt es `break`:

Python-Programm

```
if op1 == "":  
    break;
```

- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).
- Wir würden gerne (kontrolliert) die `while`-Schleife verlassen!
- Dafür gibt es `break`:

Python-Programm

```
if op1 == "":  
    break;
```

- Will man in der `while`-Schleife den nächsten Durchlauf beginnen, benutzt man `continue`:

- Man kann das Programm mit `^C` beenden (oder durch eine falsche Eingabe).
- Wir würden gerne (kontrolliert) die `while`-Schleife verlassen!
- Dafür gibt es `break`:

Python-Programm

```
if op1 == "":  
    break;
```

- Will man in der `while`-Schleife den nächsten Durchlauf beginnen, benutzt man `continue`:

Python-Programm

```
if op2 == "":  
    print("Op2 ist leer!")  
    continue
```



- Wollen wir richtige Programme schreiben, brauchen wir Werkzeuge (**Tools**).
- **Texteditor** (nicht Word!), möglichst mit integriertem **Syntaxchecker**.
- Werden Zeilen zu lang, müssen sie **umgebrochen** werden.
- **Kommentare** sind hilfreich, um das Programm zu verstehen.
- Block-, Fließtext und `doctsring`-Kommentare
- Python-Programme können auf viele verschiedene Arten **gestartet** werden.
- **IDLE** ist eine schöne und einfache **IDE** (Integrated Development Environment).
- In `while`-Schleifen gibt es `break` und `continue`.