universität
innsbruck

Brückenkurs Informatik

Algorithmik und Programmieren

Christoph Kelter, Michael Tschugnall, Philipp Zech

Organisation

Teilnahme

Zielgruppe

- Einsteiger in die Informatik
- Studierende ohne Programmierkenntnisse

Ganz locker.

- Die Teilnahme am Kurs ist freiwillig.
- Gruppenarbeit ist ausdrücklich erwünscht.
  - Helfen Sie anderen Studierenden; das hilft auch Ihnen.
  - Erst wenn Sie die Lösungen erklären können, haben Sie sie wirklich verstanden.
- Stellen Sie Fragen!
- Melden Sie sich, wenn es zu schnell oder zu langsam geht.

Organisation

Inhalt

Ablauf

- Theorieblöcke
- Praktische Übungen
  - Sie lernen am meisten, wenn Sie selbst programmieren.
  - Auch in Gruppen sollte jeder selbst programmieren!
- Schriftliche Prüfung

Ziele

- Grundlagen der Programmierung
- Probleme abstrahieren und lösen

Organisation

Ressourcen

- Termine und Kursunterlagen: OLAT
- Feedback und Fragen bitte an Philipp.Zech@uibk.ac.at.

Organisation

Python unter Linux

Wir benötigen für diesen Kurs

- eine Python-Installation (für diesen Teil des Brückenkurses)
- Linux (insbesondere für den Computer-Teil)

Organisation

Blitzumfrage

Ich habe beim ZID einen Linux-Account. . .

- **A** nicht beantragt
- **B** beantragt und erhalten, aber noch nicht ausprobiert
- **C** beantragt und erfolgreich ausprobiert
  (z.B. mit `ssh -l c...- zid-gpl.uibk.ac.at`)
- **D** anderes

Organisation

Blitzumfrage

Ich habe Linux. . .

- **A** auf meinem Rechner installiert und erfolgreich ausprobiert
- **B** auf einem bootfähigen USB-Stick installiert und erfolgreich ausprobiert
- **C** installiert aber nicht bzw. erfolglos ausprobiert
- **D** nicht installiert

Organisation

Blitzumfrage

Ich habe. . .

- **A** Zugriff auf Linux (z.B. mittels ZID-Linux-Account und `ssh` oder mittels selbst installiertem Linux)
- **B** keinen Zugriff auf Linux
- **C** keine Ahnung
- **D** anderes

Organisation

Blitzumfrage

Ich habe. . .

- **A** einen geeigneten Rechner (Laptop), auf dem ich während der Brückenkurs-Lehrveranstaltungen Programmierübungen in Python durchführen kann.
- **B** keinen solchen Rechner.
- **C** keine Ahnung.

Organisation

Erster Python-Test

1. Linux-Kommandozeile
   Unter Linux:
   - Öffnet eine *Konsole*, *Terminal*, *Shell*, ...

   Ansonsten:
   1. Öffnet euer SSH-Programm (*PuTTY*, ...)
   2. Loggt euch auf `zid-gpl.uibk.ac.at` ein.
2. Tippt in die Kommandozeile: `python --version` *Enter/Return*
   Was seht ihr:
   - `Python 3...`: OK
   - `Python 2...`: Schritt 2 wiederholen mit `python3 --version`
   - nichts dergleichen: bitte melden

Theorie 0: Was ist Python?

Theorie 0

Was ist Python?

- Python ist eine *Skriptsprache*.
  Python-Programme werden von einem *Interpreter* ausgeführt.
- Zwei Versionen sind verbreitet:
  - Python 2: obsolet
  - Python 3: neue Features, sauberer durchdacht
- Wir verwenden Python 3 in diesem Kurs.

Theorie 0

Eigenschaften von Python

- Open Source
- plattformunabhängige Skriptsprache
- einfach, verständlich und schnell zu erlernen
- dennoch sehr mächtig und von großer praktischer Bedeutung

Dies ist kein Python-Kurs. Wir werden uns auf das Notwendigste beschränken.

Theorie 0

Hilfreiche Ressourcen

- Die Python Dokumentation docs.python.org
- Online-Kurse:
  - LearnPython
  - Python for you and me
  - The Python Tutorial
- Bei konkreten Fragen: stackoverflow.com

Theorie I: Datentypen & Variablen

Theorie I

Datentypen

Python stellt Datentypen für die gängigsten Anwendungen (Built-in Types) zur Verfügung:

- *String* (Zeichenkette, Text): `"This is text!"`, `'Dies auch.'`
- *Integer* (Ganzzahlen): `1`, `2`, ...
- *Float* (Gleitkommazahlen): `1.35`, `1.56e-5` $= 1,56 \cdot 10^{-5}$
- *Bool* (Wahr/Falsch): `True`, `False`

Theorie I

Variablen

- *Literale* (wie eben benutzt) repräsentieren sich selbst.
- *Variablen* können jederzeit beliebige Werte zugewiesen werden.
- In Python muss der Typ nicht explizit benannt werden, und kann sich zur Laufzeit ändern.
- Vom (dynamischen) Typ hängt ab, welche Operationen für diese Variable zulässig sind.

Theorie I

Variablennamen

- Gültige Variablennamen sind eine beliebige Kombination aus folgenden Zeichen: a-z, A-Z, 0-9, _
- Beachten Sie folgende Sonderfälle:
  - Variablennamen dürfen nicht mit einer Ziffer beginnen.
  - Reservierte Schlüsselwörter, z.B.:
    `False`, `True`, `and`, `break`, `continue`, `def`, `elif`, `else`, `for`, `if`, `in`, `not`, `or`, `return`, `while`
- Variablen sollten *selbsterklärende* Namen tragen!

Theorie I

Ausdrücke und Anweisungen

- Ein *Ausdruck* (*expression*) hat einen Wert; eine *Anweisung* (*statement*) hat einen Effekt.
- Ausdrücke lassen sich z.B. aus Variablen und mathematischen *Operatoren* erstellen, beispielsweise +, *, <, ==, uvm.
- Eine Wertzuweisung eines Ausdrucks an eine Variable mittels des Zuweisungsoperators = stellt eine Anweisung dar.
  Beachten Sie, dass = einer Variable einen Wert zuweist, während == zwei Werte vergleicht, ohne sie zu verändern.

```
>>> x = 2 + 1
>>> x = 2 * x
>>> print(x)
6
>>> x == 5
False
>>> print(x)
```

Theorie I

Präzedenz von Operatoren

- Die Operatoren-Präzedenz bestimmt, in welcher Reihenfolge die Operatoren eines Ausdrucks angewendet werden.
- Siehe die Python-Dokumentation für eine vollständige Liste der Operatoren und ihrer Präzedenzen.

Einschub: Skripte

Programmieren in einem Editor

Bisher haben wir interaktiv direkt im Interpreter gearbeitet, gestartet mit `python` bzw. `python3`.
In der Praxis werden Sie Programme in einem *Editor* erstellen. Geben Sie Ihrer Python-Skript-Datei einen Namen, der auf `.py` endet.

Tipp

- Ein einfacher Terminal-Editor ist `nano`. Starten Sie ihn mit dem Namen Ihres Skripts als Argument: `nano myScript.py`
- Ein einfacher, graphischer Editor mit Python-Syntax-Hervorhebung ist `gedit`.
- Emacs kann alles, will aber gelernt sein.

Einschub: Skripte

Starten eines Python-Skripts

Zwei Varianten, ein Python-Script zu starten:

- Das Script abspeichern und mit `python3 myScript.py` starten
- Das Script mit einem Shebang (Magic Line) beginnen, ausführbar machen und starten

Es kann auch eine IDE verwendet werden, aber das wird für Einsteiger nicht empfohlen, da sie Nutzer von für das Verständnis Wichtigem abschirmen.

Einschub: Skripte

Starten eines Python Skripts mit Shebang

1. Im Editor das Skript mit einem Shebang beginnen:

```
1 #!/usr/bin/env python3
2 print("Hello World!")
```

2. im Terminal ausführbar machen (nur einmal):

```
1 chmod +x myScript.py
```

3. und starten:

```
1 ./myScript.py
```

Theorie I

Übungsblatt 1: Variablen

Theorie II: Kontrollfluss

Theorie II

`if`-Anweisung

- Anweisungen werden sequenziell ausgeführt.
- Ist das nicht genug, kann der Kontrollfluss gesteuert werden.
- Das `if`-Statement
  - führt einen Code Block nur aus, wenn die Bedingung `condition` erfüllt ist;
  - kann optional mit einem `else`-Block ergänzt werden,der ausgeführt wird, wenn die Bedingung nicht erfüllt ist.

```
1 if condition:
2   # executed if condition true
3 else:
4   # executed if condition false
```

Theorie II

`if`-Anweisung: Beispiel I

```
>>> if a < 0:
...     print('a is negative')
...     print('this line belongs to the same code block')
>>> print('this is a new block and is always executed')
```

- Im Gegensatz zu den meisten Programmiersprachen ist die Einrückung in Python nicht optional, sondern definiert Codeblöcke.
- Sie können Code mit <tab> oder mit 2–4 Leerzeichen einrücken, sollten aber bei einer dieser Optionen bleiben.

Theorie II

`if`-Anweisung: Beispiel II

```
>>> if a < 0:
...     print('a is negative')
>>> elif a > 0:
...     print('a is positive')
>>> else:
...     print('a must be zero then')
```

Mit dem Schlüsselwort `elif` lassen sich vielfache Alternativen kompakter und ohne Verschachtelung angeben.

Quiz

Wie sähe das obige Programmstück ohne Verwendung von `elif` aus?

Theorie II

`while`-Anweisung

Mit der `while`-Anweisung wird ein Programmblock wiederholt, solange die Bedingung `condition` erfüllt ist.

```
1 while condition:
2   # executed as long as condition is true
```

Theorie II

`while`-Anweisung: Beispiel

```
>>> i = 0
>>> while i < 3:
...     print('i is:', i)
...     i = i + 1
...
i is: 0
i is: 1
i is: 2
```

Achten Sie darauf, dass die Bedingung nicht für immer `True` ist, da die Schleife sonst endlos ist.

Theorie II

`break` and `continue`

- Eine Schleife muss nicht immer bis zum Ende ausgeführt werden.
- Mit dem Schlüsselwort `continue` wird der *aktuelle* Schleifendurchlauf abgebrochen und ggf. der folgende angetreten.
- Mit dem Schlüsselwort `break` wird die Schleife sofort *beendet*.

```
>>> i = 0
>>> while i < 1000:
...     i = i + 1
...     if i == 4: break
...     if i == 1: continue
...     print('i is:', i)
...
i is: 2
i is: 3
```

Theorie II

Übungsblatt 1: Kontrollfluss

Theorie III: Funktionen

Theorie III

Funktionen

- Mehrfach verwendete Codeteile können in Funktionen ausgelagert werden.
- Eine Funktion sollte immer genau einen Zweck erfüllen!
- Funktionen können, müssen aber keinen Rückgabewert haben (übergeben mittels `return`-Anweisung).
- Die Parameterliste `parameters` kann beliebig lang sein, also auch leer.

```
1 def functionName(parameters):
2     # function body
```

Theorie III

Funktionen: Beispiele

Die folgenden Minimalbeispiele sollen nur zeigen, wie Funktionen definiert werden können. Eine Funktion umfasst typischerweise mehr als nur eine Zeile Code.

```
>>> def mul(x, y):
...     return x * y
...
>>> mul(2, 3)
6
```

```
>>> def printABC():
...     print('abcdefghijklmnopqrstuvxyz')
...
>>> printABC()
abcdefghijklmnopqrstuvxyz
```

Theorie III

Übungsblatt 1: Funktionen

Theorie IV: Datenstrukturen

Theorie IV

Datenstrukturen

- Bisher: Einzelne Variablen und Konstanten
- Datenstrukturen erlauben es, „große“ Datenmengen zusammenzufassen und zu verarbeiten.

Theorie IV

Listen

- Die einfachste Datenstruktur in Python ist die *Liste*.
- Eine Liste ist eine geordnete, endliche Sammlung von Elementen beliebiger Datentypen.
- Achtung: *geordnet* heißt nicht zwangsläufig *sortiert*!

```
>>> list = [2,3,1,2]
>>> list
[2, 3, 1, 2]
```

Theorie IV

Kontrollfluss

- Mit einer `for`-Anweisung kann ein Programmblock für jedes Element einer Sequenz `sequence` ausgeführt werden.
- Diese `sequence` ist für uns eine Liste (Python kennt jedoch noch weitere Sequenz-Datentypen).

```
1 for variable in sequence:
2   # executed once for each element in sequence
```

Theorie IV

Kontrollfluss: Beispiel

- Mit dieser Schleife wird jedes Element der Liste `[1,2,3]` einmal ausgegeben.
- Sie können statt der Variable `i` auch jeden anderen gültigen Bezeichner verwenden.

```
>>> for i in [1,2,3]:
...     print(i)
...
1
2
3
```

Theorie IV

`range()`

Zum Iterieren über eine Folge ganzer Zahlen verwendet man typischerweise die Funktion `range()`, die einen sehr effizienten Iterator zur Verfügung stellt:

```
>>> for i in range(3):
...     print(i)
...
0
1
2
>>> for i in range(1, 3):
...     print(i)
...
1
2
```

Hinweis

Meist i.d.R. besser, über Elemente zu iterieren, als über Indizes.

Theorie IV

Zugriff auf Elemente in Listen

- Eine Liste kann Elemente verschiedener Datentypen beinhalten:

```
>>> for i in ['a', 2.3, 5]:
...     print(i)
...
a
2.3
5
```

- Einzelne Elemente oder Sublisten können auch referenziert werden:

```
>>> list = [1,2,3]
>>> list[2]
3
>>> list[1]
2
>>> list[0:2]
[1, 2]
```

Theorie IV

Listenfunktionen

- `append()` hängt ein neues Element an die Liste an.
- `+` verkettet Listen.

```
>>> list_a = ['a', 'b', 'c']
>>> list_b = ['x', 'y', 'z']
>>> list_a.append('d')
>>> list_a
['a', 'b', 'c', 'd']
>>> print(list_a + list_b)
['a', 'b', 'c', 'x', 'y', 'z']
```

Theorie IV

Listenfunktionen II

- `*n` wiederholt eine Liste `n` mal.
- `in` prüft, ob ein Element in einer Liste enthalten ist.
- `len(list)` gibt die Länge der Liste zurück.

```
>>> list_b * 3
['x', 'y', 'z', 'x', 'y', 'z', 'x', 'y', 'z']
>>> 2 in [1, 2, 3]
True
>>> len(list_a)
3
```

Theorie IV

Übungsblatt 2: Listen

Theorie IV

Zeichenketten

- *Strings* (Zeichenketten) werden in Python zwischen einfachen oder doppelten Hochkommata eingeschlossen.
- Zwischen beiden Varianten gibt es *keinen* Unterschied.
- Bei einfachen Hochkommata darf das doppelte Hochkomma im String enthalten sein, und umgekehrt.
- Viele Funktionen, die bei Listen besprochen wurden, können auch auf Strings angewendet werden: `for`, `+`, `*`, `n`, ...

```
>>> text = 'a' + "b" + "c"
>>> for c in text:
...     print(c)
...
a
b
c
>>> "abc" * 3
'abcabcabc'
```

Theorie IV

Übungsblatt 3: Strings