Programmieren mit Python

Teil 1: Einleitung

Dr. Aaron Kunert
aaron.kunert@salemkolleg.de

23. September 2021

Zu Beginn ...

Kurze Vorstellungsrunde

Schaffst Du es *in 60 Sekunden* folgende Fragen möglichst knackig und aussagekräftig zu beantworten?

- Wer bist Du?
- Windows, Mac oder Linux?
- Welche Vorkenntnisse hast Du beim Programmieren?
- Warum hast Du Dich zum Python-Kurs angemeldet?
- Wann wäre der Kurs für Dich perfekt gelaufen? (Best Case Szenario)
- Wann würdest Du den Kurs nicht weiter besuchen? (Worst Case Szenario)

Organisation des Kurses

- Ich bin die nächsten 3 Wochen verreist. D.h. nächster Termin am 21. Oktober
- Pausen: Je 20-30 Minuten zum Frühstück und einmal gegen ca. 11 Uhr. Bitte danach pünktlich kommen!
- Skript und alle Unterlagen sind im FirstClass
- Gelegentlich gibt es ein Aufgabenblatt (FirstClass) → Ca. 5 Tage Bearbeitungszeit, Abgabe per Email → individuelles Kurzfeedback
- Lernleistung durch: Anwesenheit, Mitarbeit und Bearbeitung der Aufgabenblätter
- Wissenschaftliche Arbeit ist möglich
- Kommunikation erstmal per E-Mail
- Fragen sind immer und über alle Kanäle willkommen!

Didaktik des Kurses

- Mischung aus Vortrag, Präsenzübungen und Live-Coding
- Lösungen der Präsenzübungen gibt's im Handout im Firstclass
- Achtung: Präsenzübungen können erstmal frustrierend sein.
- Im Idealfall: Mehr Praxis statt Erklärungen

Ziele des Kurses

- Einblick in die „Denkweise“ eines Computers
- Einige universelle Konzepte von Programmiersprachen kennenlernen
- Schulung des analytischen Denkens
- Verständnis von Python-Syntax
- Programmierung eines rudimentären Quizspiels

Wo findet man Hilfe/Infos?

- Google
- `stackoverflow.com`
- Youtube (z.B. Tutorials)
- `docs.python.org/3`
- Bücher (z.B. *Python Crashkurs* v. Eric Matthes)
- `mailto: aaron.kunert@salemkolleg.de`

Was ist Python?

Wie funktioniert überhaupt die Programmierung in Python?

1. Man **schreibt** eine Abfolge von Befehlen/Anweisungen in eine Text-Datei (nicht Word!)
2. Danach lässt man diese Datei vom Python-Interpreter **ausführen**.

Was wird benötigt?

Am Anfang

- Compiler/Interpreter
- Texteditor (z.B. Mac: Xcode, Windows: Edit)

Später

- Google
- Integrierte Entwicklungsumgebung (IDE)
- Versionskontrolle (VCS)
- Virtueller Maschinen
- Datenbanken
- Grafikbearbeitung

Editor und Compiler müssen nicht auf dem eigenem Computer installiert sein. Es gibt dafür auch cloudbasierte Lösungen.

Warum Python?

- Einfaches Setup
- Einstiegsfreundliche Syntax
- Python ist eine Hochsprache
- Python muss nicht kompiliert, sondern nur interpretiert werden
- Große Community → großes *Ecosystem*
- Python ist extrem vielseitig
- Python ist plattformunabhängig

Typische Einsatzbereiche

- Automatisierung
- Webscraping
- Datenanalyse
- Webentwicklung

Wie Programmierer denken

Wie lernt man analytisches Denken?

Everyone in this country should learn to program a computer, because it teaches you to think.

(Steve Jobs)

Phasen des Lernens einer Programmiersprache

1. Annäherung: Fokus auf dem Begreifen der Grundkonzepte
2. Syntax: Fokus auf der korrekten Anwendung der Syntax
3. Funktionalität: Fokus liegt darauf, Problemstellungen *pragmatisch* zu lösen
4. Design: Fokus auf les-und wartbaren Code
5. Architektur: Fokus auf Strategie, Projekte nachhaltig und erweiterbar umzusetzen

Problem Solving

Sobald man die Syntax korrekt verwenden kann, steht das Lösen von Problemen beim Programmieren im Fokus.

Dabei ist die Kunst nur wenige, klare begrenzte Bausteine (die Befehle der Sprache) *kreativ* so zusammenzusetzen, damit das gegebene Problem gelöst wird.

Problemlösungsstrategien

- **Trial** and Error
- Formuliere laut und möglichst präzise, was eigentlich die Problemstellung ist
- Zerlege das Problem in kleinere Probleme oder mach Dir Zwischenziele
- Gibt es schon eine ähnliches Problem, was Du gelöst hast und von wo aus Du starten kannst?
- Erkläre anderen das Problem und was Du schon bisher geschafft hast
- Ändere Dein Denken: Scheitern ist nicht das Ende des Weges, sondern der Anfang
- To be continued

Die Konsole

Das Sprachrohr zum Computer

Definition: Konsole

Die Konsole ist ein simples Programm, das nur aus einem Eingabefeld besteht, und mit dem man mit einem anderen (in der Regel komplexeren Programm) mittels spezifischer Befehle kommunizieren kann.

Beispiele

- Windows-Eingabeaufforderung (Kommunikation mit Windows)
- Mac-Terminal (Kommunikation mit MacOs)
- Browser-Konsole
- Die Python-Konsole

Für Programmiererinnen ist die Konsole der wichtigste Kommunikationsweg zu ihrem Computerprogramm.

Übung

Überprüfe, ob Python bei Dir installiert ist

1. Google wie man die Konsole bzw. das Terminal zum Betriebssystem öffnet
2. Öffne die Konsole
3. Prüfe, ob Python installiert ist, indem Du einen der folgenden Befehle ausprobierst
   - `python --version`
   - `python3 --version`
4. Interpretiere die Antwort

Programmieren in der Cloud

Schnell und unkompliziert einsteigen

Browserbasierte IDE verwenden

1. Gehe auf https://replit.com
2. Erstelle ein Konto (Sign up)
3. Klicke auf „Create repl“
4. Wähle als Template „Python“ aus

Erste Schritte im REPL

(Read-Evaluate-Print-Loop)

Probier mal folgende Kommandos aus

- `3 + 4`
- `2 - 7`
- `"Hello" + "World"`

Übung

Was machen die folgenden *Operatoren*?

- `+`
- `-`
- `*`
- `/`
- `**`

Und diese?

- `%`
- `//`
- `==`
- `<=`
- `<`

Lösungen

Operatoren I

Die Operatoren + und - sind klar. Die Operatoren * und / bezeichnen Multiplikation und Division. Der Operator ** berechnet die Potenz (hochnehmen).

Operatoren II

Der Operator % ist der Modulo-Operator (vgl. Wikipedia). Der Operator // arbeitet analog zur Divison, rundet das Ergebnis jedoch auf die nächste ganze Zahl ab. Die Operatoren == (Gleichheit), <= (Kleinergleich), < (kleiner als) sind Vergleichsoperatoren und geben entweder `True` oder `False` zurück

Übung

Wie rechnet Python?

- Wird Punkt-vor-Strich berücksichtigt?
- Kann man mit Klammern die Reihenfolge beeinflussen?
- Was ist der Unterschied zwischen `10/5` und `10//5` ?
- Was bedeutet das Kommando _?
- Wie kann man Zwischenergebnisse in Variablen speichern?

Lösung

Rechenregeln

- Python rechnet Punkt-vor-Strich.
- Python berücksichtigt Klammern.
- Das Ergebnis von `/` ist stets eine Fließkommazahl, das Ergebnis von `//` ist stets eine ganze Zahl.
- Das Kommando `_`, referenziert das vorherige Ergebnis.
- Zwischenergebnise lassen sich mittels des Zuweisungsoperators `=` in einer Variable speichern.

Variablen

Jeder Wert in Python kann in einer Variable gespeichert werden:

```
my_variable = 3
```

Die Zuweisung darf auch das Ergebnis einer Berechnung sein:

```
my_new_variable = 3 + 5
```

Die Zuweisung darf auch weitere Variablen enthalten:

```
my_brand_new_variable = my_variable + my_new_variable
```

Man darf auch Kettenzuweisungen machen:

```
a = b = c = 100
```

Gültige Variablennamen

- Erlaubt sind Buchstaben (nur ASCII), Ziffern und Unterstriche
- Der Name darf nicht mit einer Ziffer starten
- Beliebige Länge
- Wer's schon kennt als *regulärer Ausdruck*: `[_a-zA-Z][_0-9a-zA-Z]*`
- Schlüsselwörter sind nicht erlaubt

Liste der Schlüsselwörter

`False`	`None`	`True`	`and`	`as`
`await`	`break`	`class`	`continue`	`def`
`else`	`except`	`finally`	`for`	`from`
`import`	`in`	`is`	`lambda`	`nonlocal`
`pass`	`raise`	`return`	`try`	`while`
`assert`	`global`	`with`	`elif`	`or`
`del`	`not`	`async`	`if`	`yield`

Style-Guide Variablennamen

- Englische Wörter
- Nur Kleinbuchstaben
- Möglichst aussdrucksstarke Namen verwenden
- Keine Angst vor langen Namen
- Namen, die aus mehreren Worten bestehen, mit Unterstrich trennen (*snake-case*)

z.B. `students_in_this_room`, `number_of_unpaid_bills`

Übung

Probier's aus!

- Welchen Wert hat eine Variable, wenn man sie nicht vorher definiert hat?
- Was passiert, wenn man eine Variable definiert, die schonmal verwendet wurde?
- Wie kann man eine Variable mit Wert `3` um `1` vergrößern?

Lösung

- Verwendet man eine undefinierte Variable, wird ein Fehler geworfen
- Ja, man kann eine Variable einfach neu definieren
- Hat beispielsweise `my_variable` den Wert 3, so lässt sich der Wert wie folgt vergrößern: `my_variable = my_variable + 1`

Datentypen

Jeder Wert in Python hat einen *Datentyp*. Unter anderem gibt es folgende *primitive* Typen in Python.

- `int` Integer (ganze Zahlen)
- `float` Float (Dezimalzahlen)
- `bool` Boolean (Wahrheitswerte)
- `str` String (Zeichenketten)
- `NoneType` (Typ des leeren Werts `None`)

Integer

Ganze Zahlen wie z.B. `1`, `-1`, `0`. Nicht aber `2.0` oder `0.0`.

Float

Fließkommazahlen, z.B. `3.1415925`. Achtung: Bei Float-Berechnungen können schnell „Überraschungen“ auftreten: Was ergibt z.B. `1.2 - 1.0` ?

Boolean

Booleans sind eine Sonderform von `int` und können nur die Werte `True` (entspricht 1) und `False` (entspricht 0) annehmen. Sie entstehen in der Regel, wenn man Fragen im Programm stellt (z.B. `3 < 4` oder `1 == 2`).

String

Strings sind beliebige Zeichenketten und müssen in (ein-, zwei- oder dreifache) Anführungszeichen eingeschlossen werden. Die Ausdrücke `'hello'`, `"Hello"` und `"""Hello"""` sind (fast) äquivalent.

Mehrzeilige Strings

Ein *Stringliteral* kann nur innerhalb einer Zeile definiert werden. Soll ein String mehrere Zeilen umfassen, müssen dreifache Anführungszeichen verwendet werden.

Steuerzeichen

Gewisse Kombinationen mit Backslash sind reservierte Steuerzeichen. So bezeichnet beispielsweise `\n` einen Zeilenumbruch und `\t` ein Tabulatorzeichen.
Beispiel: `"This text\nfills two lines"`

Escaping

Möchte man ein Steuerzeichen nicht ausführen, sondern buchstäblich nehmen. Muss man sie mit einem Backslash *escapen* bzw. maskieren.
Beispiel: `"This text fits in\\n one line"`

Raw-Strings

Möchte man alle Steuerzeichen eines Strings ignorieren, kann man ihn als *Raw-String* definieren.
Beispiel: `r"This \n String \t has no control characters"`

Typecasting (Umwandlung von Typen)

Implizit

Bei manchen Operationen nimmt Python automatisch eine Typumwandlung vor.
Beispiel: `1 + 2.0` ergibt `3.0`

Explizit

Die Funktionen `int()`, `float()`, `str()` und `bool()` führen jeweils eine Typumwandlung durch (sofern möglich). Beispiele:

- `int(2.0)` ergibt `2`
- `float(2)` ergibt `2.0`
- `int("3")` ergibt `3`

Typ einer Variablen ermitteln

Mit der Funktion `type()` lässt sich der Typ bestimmen, z.B. `type(3.2)`.

Übung

Versuche die Fragen erst ohne Python zu beantworten, überprüfe Deine Vermutung

- Welchen Datentyp hat das Ergebnis von `3 - 1.0` ?
- Was ist das Ergebnis von `"2" + 1` ?
- Was ist das Ergebnis von `"2" + "2"`?
- Sind die beiden Werte `0` und `"0"` gleich?
- Sind die beiden Werte `2` und `True` gleich?
- Sind die beiden Werte `bool(2)` und `True` gleich?
- Sind die beiden Werte `1` und `True` gleich?

Lösungen

Typaufgaben

- Der Datentyp des Ergebnisses ist `float`
- Fehlermeldung
- Das Ergebnis ist `"22"`
- Nein
- Nein
- Ja
- Ja

Übung

Erkläre mit Deinen eigenen Worten

- Nach welcher Regel wandelt `int()` eine Fließkommazahl in eine ganze Zahl um?
- Nach welchen Regeln wandelt `bool()` Zahlen und Strings in einen Wahrheitswert um?

Operatoren

Die wichtigsten Operatoren

- `+` (Addition oder Zusammenkleben von Strings)
- `-` (Subtraktion)
- `*` (Multiplikation)
- `/` (Division, ergibt immer ein Wert vom Typ `float`)
- `**` (Potenzierung)
- `%` (*modulo-Operator*: Rest bei ganzzahliger Division)
- `//` (Division und Abrunden, ergibt immer ein Wert vom Typ `int`)
- `==` (Vergleichsoperator, ergibt immer ein Wert vom Typ `bool`)
- `!=` (Ungleichheitsoperator, ergibt das Gegenteil von `==`)

Operator-Präzedenz

1. Klammern
2. **
3. *, /, //, %
4. +,-

Operatoren gleichen Rangs werden innerhalb eines Ausdrucks von links nach rechts abgearbeitet.

Ausnahmen:
Potenzierung (**) und Zuweisung (=) werden von rechts nach links verarbeitet.

Kombinierte Zuweisung

Oft möchte man eine gegebene Variable neu zuweisen:

```
counter = 1
counter = counter + 1          # counter = 2
```

Dies lässt sich auch kurz schreiben als

```
counter = 1
counter += 1          # counter = 2
```

Analog sind die Operatoren `-=`, `*=`, `/=`, etc. definiert.