Grundkurs: Programmieren

Einführung in grundlegende Programmierkonzepte mit Python

Christoph Sonntag

WS 17/18

Universität Passau

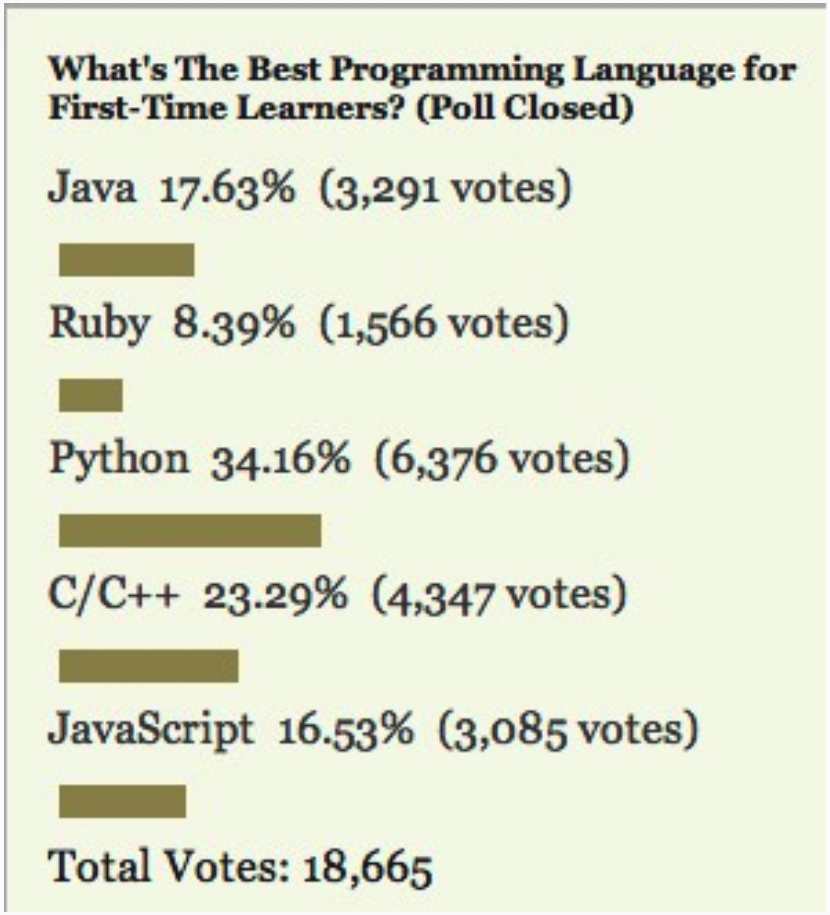
Einführung in die Programmierung

Erwartungen und Vorkenntnisse

- Erwartungen an den Kurs?
- Bereits Programmierkenntnisse aus Schule/Universität?
- Kursziele
  - grundlegendes Verständnis
  - “mit Informatikern reden können”
  - Angst nehmen

Die Programmiersprache Python

- Warum Python?
  - flache Lernkurve, sehenswerte Ergebnisse bereits nach dem ersten Tag
  - verankert in Forschung und Wirtschaft
  - der englischen Sprache sehr ähnlich



```
1  languages = ["C", "C++", "Java", "Python", "Fortran"]
2  modern_languages = \
3      list((x for x in languages if x is not "Fortran"))
```

Installieren von Python

- Python 3.6.3 unter `https://www.python.org/downloads/` herunterladen und ausführen
- Zum ‘PATH’ hinzufügen und ‘. . . for all users’ deaktivieren

Hello World

```
1    print("Hello World!")
```

- IDLE suchen und starten
- eintippen und mit Enter ausführen
- gibt den Text (String) ‘Hello World!’ auf der Konsole aus

Glückwunsch
Ihr habt gerade euer erstes Code-Fragment geschrieben und ausgeführt!

Hello World

Aufgabe 1
Erweitere das Programm so, dass der `String` ‘Hello World’ 6-mal auf der Konsole ausgegeben wird.

Hello World

Aufgabe 1
Erweitere das Programm so, dass der `String` ‘Hello World’ 6-mal auf der Konsole ausgegeben wird.

Lösung

```
1 print("Hello World")
2 print("Hello World")
3 print("Hello World")
4 print("Hello World")
5 print("Hello World")
6 print("Hello World")
```

Hello World

- `sleep(num)`
  - stoppt die Ausführung des Programms für `num` Sekunden
  - ist nicht Teil des Standard-Pythons, sondern muss importiert werden. `from time import sleep`

Hello World

- `sleep(num)`
  - stoppt die Ausführung des Programms für `num` Sekunden
  - ist nicht Teil des Standard-Pythons, sondern muss importiert werden. `from time import sleep`

Aufgabe 2
Lass das Programm einen realistischen Monolog ausführen.

Verwende hierzu `sleep`

Hello World

- `sleep(num)`
  - stoppt die Ausführung des Programms für `num` Sekunden
  - ist nicht Teil des Standard-Pythons, sondern muss importiert werden. `from time import sleep`

Aufgabe 2
Lass das Programm einen realistischen Monolog ausführen.

Verwende hierzu `sleep`

Lösung

```
1 from time import sleep
2
3 print("Hey, it's James!")
4 sleep(2)
5 print("I'm working on a ChatBot right now")
6 sleep(3)
7 ...
```

Programmiersprachen

- Unterscheidungsmerkmale
  - Programmierparadigma: imperativ, funktional oder objektorientiert
  - Typsicherheit
  - kompiliert vs. interpretiert
  - allgemein vs. domänenspezifisch
  - hardwarenah vs. höhere Programmiersprachen

Programmiersprachen

- Imperative Programmiersprachen: C/C++, C#, Java ...
- Funktionale Programmiersprachen: SQL, Haskell, Erlang, (Scala) ...
- Objektorientierte Programmiersprachen: C++, C#, Java, Javascript, PHP, Python ...

Imperative Sprachen (C/C++, C#, Python, Java, . . .)

- ältestes Programmierparadigma
- große Verbreitung in der Industrie
- besteht aus Befehlen (lat. imperare = befehlen)
- Abarbeiten der Befehle ‘Schritt für Schritt’
- sagt einem Computer, ‘wie’ er etwas tun soll

```
1 print("Hey, whats' up?")
2 sleep(3)
3 print("Learning Python right now")
4 sleep(2)
```

- Verwendung
  - ‘Standard-Software’, hardwarenahe Entwicklung

Funktionale Sprachen (Haskell, Erlang, SQL, Lisp, . . .)

- vergleichsweise modern
- sagt einem Computer, ‘was’ das Ergebnis sein soll
- SELECT name FROM students WHERE major=‘law’ AND semester=‘1’;
- Verwendung
  - akademische Zwecke
  - sicherheitskritische und . . .
  - hoch performante Anwendungen

```
1 square :: [Int] -> [Int]
2 square a = [2*x | x <- a]
```

```
x = x + 1
```

Objektorientierte Sprachen (Java, Python, C++, C#, . . .)

- starke Verbreitung
- Abbilden der realen Welt der Dinge auf Objekte
- Klasse: Bauplan eines Objekts bestehend aus Eigenschaften (Attributen) und Methoden
- Vererbung möglich
- Verwendung
  - Standard-Software
  - Modellierung realer Projekte(Unternehmen, Mitarbeiter, Kunden, Waren, . . .)
  - große Projekte ($\rightarrow$ Planung durch Klassendiagramme)

Objektorientierung: Beispiel

```
1  class Konto:
2      def __init__(self, name, nr):
3          self.inhaber = name
4          self.kontonummer = nr
5          self.kontostand = 0
6      def einzahlen(self, betrag):
7          self.kontostand = kontostand + betrag
8      def auszahlen(self, betrag):
9          self.kontostand = kontostand - betrag
10     def ueberweisen(self, ziel, betrag):
11         ziel.einzahlen(self.betrag)
12         self.auszahlen(betrag)
13     def kontostand(self):
14         return self.kontostand
15
16 class Unternehmenskonto(Konto):
17     def erhalteBonus(self, bonus):
18         self.kontostand = kontostand + bonus
```

Kompilierte und Interpretierte Sprachen

- kompilierte Sprachen (Java, C/C++, C#, . . .):
  - Übersetzung des (kompletten) Programmcodes in Maschienencode
  - dann Ausführung des Maschinencodes
- interpretierte Sprachen (Python, Lisp, PHP, JavaScript, . . .):
  - Übersetzung einer einzelnen Programmanweisung
  - Ausführung dieser Anweisung
  - Übersetzung der nächsten Anweisung

Hardwarenahe und höhere Sprachen

- hardwarenah: abhängig von der Bauweise des Prozessors
- höhere Sprachen: von der Bauweise abstrahiert (`print()`, `sleep()`)

```
1  .START ST
2    ST: MOV R1,#2
3        MOV R2,#1
4    M1: CMP R2,#20
5        BGT M2
6        MUL R1,R2
7        INI R2
8        JMP M1
9    M2: JSR PRINT
10       .END
```

```
1
2  a = 2;
3  i = 1;
4  # compare i ==
       20
5  # if True, jump
       to M2
6  a = a*i;
7  i++;
8  # jump to M1
9  print(a)
```

```
1  a = 2;
2  for i in range
       (1, 20) {
3
4
5      a = a*i;
6  }
7
8
9  print(a);
```

Populäre Programmiersprachen

- C++
  - imperativ, objektorientiert, typsicher, kompiliert, allgemein, höhere Sprache (dennoch hardwarenah)
  - große Verwendung in hocheffizienten Systemen (Betriebssysteme, Grafikberechnungen, Computerspiele, . . .)
  - Erweiterung von C mit Objektorientierung
- Java
  - imperativ, objektorientiert, typsicher, kompiliert, allgemein
  - im bayrischen Lehrplan und an vielen Universitäten ‘erste’ Sprache
  - ebenfalls große Verbreitung
- Python
  - (imperativ), (funktional), objektorientiert, dynamisch getypt, interpretiert, allgemein
  - große Verbreitung auch gerade im akademischen Umfeld, Web, Machine Learning und Data Science

Typen und Variablen

Wir haben ganz unbewusst bereits zwei Datentpen benutzt

- `String, str`:

```
1 print("Ich bin vom Typ String, eine Reihe von
      Zeichen")
```

- `Integer, int`:

```
1 sleep(3)    # 3 ist ein Integer
```

Typen und Variablen

- Wahrheitswert (Boolean, bool):

```
1 wahr = True
2 falsch = False
```

- Gleitkommazahlen (Float, float):

```
1 pi = 3.1415
```

- In Python gibt es sogar komplexe Zahlen `complex`:

```
1 a = complex('3+5j')
```

Typen und Variablen

- Zuweisung von Variablen mit dem Zuweisungsoperator =

```
1 a = 5
2 b = 3.14
3 c = "Hallo Grundkurs:Programmieren"
```

- mehrfache Zuweisung

```
1 a, b, c = 5, 3.14, "Hallo Grundkurs:Programmieren"
2 x = y = z = 42
```

Typsicherheit

- streng-getypte Sprachen: **Java**, C/C++, ...

```
1 int x = 42
2 float pi = 3.14
3 String greeting = "Hallo Grundkurs:Programmieren"
4 boolean truth = true
```

- dynamisch (schwach) getypte Sprachen: **Python**, Javascript

```
1 x = 42
2 pi = 3.14
3 greeting = "Hey there"
4 truth = True
```

Typsicherheit

Aufgabe
Gib folgende Ausdrücke in den Python Interpreter ein:

```
1 >>> 3 + 3.14
2 >>> "Mein Alter: " + 5
3 >>> True + 1
```

Typsicherheit

Aufgabe
Gib folgende Ausdrücke in den Python Interpreter ein:

```
1 >>> 3 + 3.14
2 >>> "Mein Alter: " + 5
3 >>> True + 1
```

True + 1 == 2?
Intern werden die Keywords `True` und `False` auf die Werte `1` und `0` vom Typ `int` abgebildet.

Typen und Variablen

Die `input()` Funktion nimmt Benutzereingaben auf der Kommandozeile entgegen und gibt sie zurück. Du brauchst kein zusätzliches Modul importieren.

Aufgabe
Lasse dich von deinem Programm begrüßen, indem du mit `input` deine Eingabe in einer Variable speicherst.

Typen und Variablen

Die `input()` Funktion nimmt Benutzereingaben auf der Kommandozeile entgegen und gibt sie zurück. Du brauchst kein zusätzliches Modul importieren.

Aufgabe
Lasse dich von deinem Programm begrüßen, indem du mit `input` deine Eingabe in einer Variable speicherst.

Lösung

```
1 >>> name = input()
2 'Christoph'
3 >>> print("Hallo " + name)
```

Entwicklungsumgebung einrichten

Achtung

Word, TextEdit, Notepad, oder Wordpad sind Textverarbeitungsprogramme, keine Quelltext-Editoren und schon gar keine Entwicklungsumgebungen

Entwicklungsumgebung einrichten

Achtung
Word, TextEdit, Notepad, oder Wordpad sind Textverarbeitungsprogramme, keine Quelltext-Editoren und schon gar keine Entwicklungsumgebungen

- Editoren wie `Sublime Text`, `Atom` oder `IDLE` sind für uns ausreichend
- große IDE's wie `Eclipse`, `IntelliJ` oder `PyCharm` bieten weitere Funktionen

Entwicklungsumgebung einrichten

- Pythonprogramme in IDLE schreiben und ausführen
  1. `Datei` > `Neue Datei`
  2. geeigneten Speicherort aussuchen, bspws. `Dokumente/GrundkursProgrammieren/helloworld.py`
  3. Programm schreiben...
  4. Programm unter `Run` > `Run Module` ausführen oder F5 drücken