mentoring

Freie Universität Berlin

Python Crashkurs

Mentoring – SoSe 2019

Anja Wolffgramm
Freie Universität Berlin
Institut für Informatik

12. April 2019

Inhalt

Inhalt

Was ist ein Algorithmus?

Definition:

- Eine Rechenvorschrift zur Lösung eines Problems,
- mit definierter Ein- und Ausgabe,
- bestehend aus wohldefinierten Einzelschritten,
- kann als Kombination natürlicher Sprache und mathematischen Formeln (sog. Pseudocode) beschrieben werden.

Eigenschaften von Python

- leicht erlernbar
- dynamische Typstrukturen: der Datentyp einer Variable wird zur Laufzeit festgelegt und kann während der Ausführung verändert werden
- der Datentyp wird aus dem ihr enthaltenen Wert abgeleitet
- Referenz-Symantik: ein Ausdruck wird zu einem Objekt ausgewertet, dessen Speicheradresse in einer Variablen-Adresse gespeichert wird

Quelle: Esponda Agüero (2012)

Inhalt

Kommandozeile

	Linux	Mac	Windows
Kommandozeile (Terminal) öffnen	`[Strg]+[Alt]+[T]` oder "Terminal" suchen	`[cmd]+[Space]` oder "Terminal" suchen	Start → `cmd`
Verzeichnisse anzeigen	`ls`		`dir`
Verzeichnis wechseln	`cd` mein/pfad/datei		`cd` mein\pfad\datei
Ins übergeordnete Verzeichnis wechseln	`cd ..`		`cd..`

Table 1: Befehle der Kommandozeile. Quellen: o.A. (2018) und Oberneder (2018)

Inhalt

Python installieren

Bitte zu Hause installieren.

Linux	Mac / Windows
Kommandozeile öffnen `[Strg]+[Alt]+[T]`	Datei herunter laden: https://www.python.org/downloads
`sudo apt install libreadline-dev` `sudo apt-get install python3`	Doppelklick und installieren

Table 2: Python installieren (Minimäxchen, 2019)

Tipp

`libreadline-dev` sorgt dafür, dass die Pfeiltasten in der Shell funktionieren (Epp, 2012).

Python starten

Python wird wie folgt in der Konsole gestartet:

```
anja@Kometes:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 1: Starten von Python3 in der Konsole. Quelle: eigene Bearbeitung

Inhalt

Python ausprobieren

Nun könnt ihr ein bisschen herum probieren:

```
1  3+7
2  5/3
3  5//3
4  9%7
5  list(range(1,10))
6  A = list(range(10,20))
7  A[0]
8  A[5:9]
9  [1,2]+[4,5]
10 # dies ist ein Kommentar
11 """ dies ist ein
12     mehrzeiliger
13     Kommentar """
14 quit()
```

Source Code 1: Ausdrücke in Python

Fragen: Was machen die Ausdrücke?

- Addition
- Division
- ganzzahlige Division
- Modulo
- erstellt eine Liste
- gibt das 1. Elem. aus
- gibt einen Bereich aus
- Konkatenation
- Kommentare

- Verlassen von Python

Inhalt

Texteditor öffnen:

Linux	Mac	Windows
Kate, Atom, GEdit	TextMate, TextWrangler	Notepad++

Table 3: Bekannte Texteditoren zum Programmieren

Vorteile guter Texteditoren:

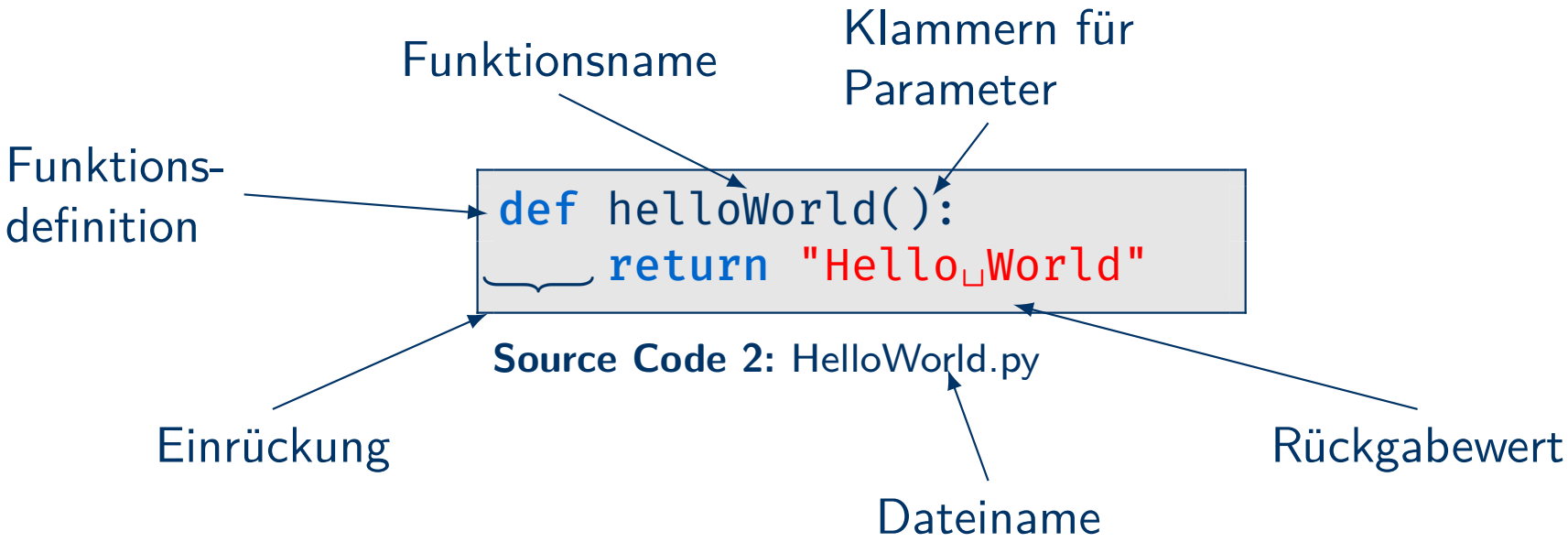
- Syntax-Highlighting
- Vorschläge für Befehle oder verwendete Namen
- unterstützt Einrücken
- multiple Tabs
- geteiltes Fenster
- Mini-Übersicht des Quellcodes

1. Programm schreiben

Erstellen einer Python-Datei:

1. **Möglichkeit:** Rechtsklick > neue Datei > HelloWorld.py
2. **Möglichkeit:** Texteditor öffnen > `[Strg] + [S]` > Namen eingeben > speichern

Code einfügen und speichern:



Source Code 2: HelloWorld.py

Programm ausführen

- Kommandozeile (Terminal) aufrufen
- Folgende Kommandos eingeben:

```
1 cd mein/pfad
2 python3
3 import HelloWorld # Dateiname
4 HelloWorld.helloWorld() # Datei.Funktionsname
```

Source Code 3: Ausführen des Programms

- Bei Änderung des Codes, muss das Programm erneut geladen werden.

```
1 from importlib import reload # einmalig
2 reload(HelloWorld) # immer, wenn etwas geändert wurde
3 HelloWorld.helloWorld() # Funktionsaufruf
```

Source Code 4: Neuladen des Programms

Inhalt

Boolean

Vergleichsoperatoren (binär)	
==	Gleichheit (infix)
!=	Ungleichheit (infix)
Logische Operatoren (unär)	
not	Negation (unär)
or	Oder (infix)
and	Und (infix)

Table 4: Boolesche Operatoren

Beispiele:

```
1  x = True # Zuweisung
2  y = False
3  x == y # Vergleich
4  x or y
5  not y # Negation
6  type(x) # <type 'bool'>
7  help(bool) # Manual, q
       beendet den Modus
```

Source Code 5: Boolesche Operatoren

Aufgabe A

Schreibe eine Funktion, die drei boolesche Parameter übergeben bekommt und nur dann `True` ausgibt, wenn die ersten beiden Parameter gleich sind und sich vom letzten unterscheiden:

$$f(a,b,c) = \begin{cases} \text{True}, & a = b = \text{True} \wedge c = \text{False} \\ \text{True}, & a = b = \text{False} \wedge c = \text{True} \\ \text{False}, & \text{sonst} \end{cases} \qquad \forall a, b, c \in \{\text{True}, \text{False}\}$$

Achtung! Lösung:

```
1 def decide(a,b,c):
2     return a==b and a!=c
```

Source Code 6: Lösung 1 zur Aufgabe A

```
1 def decide2(a,b,c):
2     return (a and b and not c) or (not a and not b and c)
```

Source Code 7: Lösung 2 zur Aufgabe A

Vergleichsoperatoren (binär)	
==	Gleichheit
!=	Ungleichheit
<	Kleiner als
>	Größer als
<=	Kleiner gleich
>=	Größer gleich

Arithm. Operatoren (unär)	
-	kehrt das Vorzeichen um
+	ändert nichts

Table 5: Integer-Operatoren

Arithm. Operatoren (binär)	
+	Addition
-	Subtraktion
*	Multiplikation
**	Power
//	ganzzahlige Division
%	Modulo

Table 6: Integer-Operatoren

```
1 x = 42 # Zuweisung
2 -x # Negation
3 type(x) # <type 'int'>
```

Source Code 8: Integer-Operatoren

Aufgabe B

Nadja kauft sich einen Taschenrechner. Dieser hat aber eine Fehlfunktion: Anstelle einen Winkel zwischen 0° und 360° auszugeben, gibt er zwar das richtige Ergebnis – allerdings viel zu groß. Schreibe eine Funktion, die das Ergebnis des Taschenrechners bekommt und dieses auf den Wertebereich $[0, 360)$ normiert. Beispiel:

$$\text{winkel}(450) \mapsto 90$$
$$\text{winkel}(45) \mapsto 45$$

Achtung! Lösung:

```
1 def winkel(w):
2     return w % 360
```

Source Code 9: Lösung zur Aufgabe B

Float

Float hat die gleichen Operatoren wie Integer und zusätzlich noch:

Arithm. Operator (binär)	
/	Division

Table 7: Float-Operator

```
1 x = 3.14152 # Zuweisung
2 x < x**2 # Vergleich
3 type(x) # <type 'float'>
```

Source Code 10: Float-Operatoren

Complex hat weniger Operatoren als Float:

Vergleichsperatoren	
==	Gleichheit
!=	Ungleichheit
Arithm. Operatoren (unär)	
-	kehrt das Vorzeichen um
+	ändert nichts
Arithm. Operatoren (binär)	
+	Addition
-	Subtraktion
*	Multiplikation
**	Power
/	Division

Table 8: Complex-Operatoren

```
1 c = 2.5+3j # Zuweisung
2 c + (4+5j)
3 type(c) #<class 'complex
      '>
```

Source Code 11: Complex-Operatoren

Liste

Listen sind veränderbare Sammlungen von Objekten verschiedener Datentypen. Sie sind als dynamische Arrays implementiert.

Vergleichsperatoren (Länge)	
==	Gleichheit
!=	Ungleichheit
<,>	Kleiner/größer als
<=, >=	Kleiner/größer gleich
Listenoperatoren	
+	Verkettung
*	Wiederholung
[i]	i-te Stelle
[i:j]	Teilliste i bis j-1
in	Tets auf Enthaltensein
not in	Test auf Nichtenthaltensein

Table 9: Listen-Operatoren

```
1 a = [] # leere Liste
2 b = [1] # 1elementige L.
3 a = b+[2,3,4]
4 1 in a
5 type(a) #<class 'list'>
6 c = a
7 a[0] = '42' # Zuweisung
8 c # gibt c aus
9 4*[True]
```

Source Code 12: Listen-Operatoren

Funktionen & Methoden für Listen	
`len(list)`	gibt die Länge aus
`max(list)`	findet das Maximum
`min(list)`	findet das Minimum
`sorted(list)`	gibt die sortierten Elemente aus
`print(list)`	gibt die Liste aus
`list.append(x)`	fügt hinten an
`list.pop()`	löscht letztes Element
`list.insert(i,x)`	Fügt x an Pos. i ein.
`list.remove(x)`	löscht das Elem. x
`list.index(x)`	Gibt den Index des 1. Matches aus
`list.copy()`	Kopiert die Liste

Table 10: Funktionen & Methoden für Listen

Quelle: (o.A., 2019)

Um durch eine Liste zu laufen, kann eine `for`-Schleife verwendet werden:

```
1 for n in [1,2,3,4]:
2     print(n) # n nimmt jeden Wert aus [1,2,3,4] an
```

Source Code 13: For-Schleife

```
1 liste = [1,2,3,4] # len(liste) = 4
2 for i in range(0, len(liste)): # 4 ist exklusive
3     print(liste[i]) # i nimmt jeden Wert von 0 bis 3 an
```

Source Code 14: For-Schleife mit Index als Zähler

Aufgabe C

Schreibe eine Funktion **mul**, die eine Liste von Zahlen erhält, jeden Wert quadriert und die veränderte Liste ausgibt. Beispiel:

$$\text{mul}([1, 2, 3, 4]) \mapsto [1, 4, 9, 16]$$

Achtung! Lösung:

```
1  def mul(liste): # Lösung 1
2      for i in range(len(liste)):
3          liste[i] = liste[i]**2
4      return liste # ein return ist eigentl. unnötig

6  def mul2(liste): # Lösung 2
7      neueListe = []
8      for n in liste:
9          neueListe.append(n**2)
10     return neueListe
```

Source Code 15: Lösung 1 und 2 zur Aufgabe C

List-Comprehension bietet eine elegante Weise, Listen zu erstellen:

```
1 [x for x in range(10)] # alle ganzen Zahlen von 0 bis 9
2 [x**2 for x in range(30) if x%2 == 0] # Quadrat aller
      geraden Zahlen
3 [x+y for x in ['ALP','MafI'] for y in ['␣1','␣2','␣3']] #
       konkateniert jedes x mit jedem y
```

Source Code 16: List-Comprehension

Quelle: (o.A., 2019)

Tupel

Tupel haben fast alle Operatoren und Funktionen wie Listen. **Achtung!** Tupel sind nicht veränderbar (immutable).

```
1  t = () # leeres Tupel
2  u = (1,) # Tupel mit einem Elem.
3  v = (2,3)
4  u + v # Konkatenation
5  v[0]
6  v[0] = 1 # Zugriffsfehler
7  'x' in v
8  type(v) # <class 'tuple'>
9  tuple(2*x for x in range(5))
```

Source Code 17: Tupel-Operatoren

String

Strings haben fast alle Operatoren und Funktionen wie Listen und Tupel.
Achtung! Auch Strings sind nicht veränderbar (immutable).

```
1  s = "spiegel"
2  s + "ei"
3  s[3]
4  s[2:5] # die 5-te Stelle
       ist exklusive
5  len(s)
6  "ABC" == 'ABC'
7  3*"ACDC"
8  print("hi\ndu")
9  type(s) # <class 'str'>
```

Source Code 18: String-Operatoren

```
1  "PyThOn".lower()
2  "PyThOn".upper()
3  "C".join(['A','D',''])
4  "Aminosäuren".find("no")
5  "hi␣du".replace("␣","\n")
```

Source Code 19: String-Methoden

Quelle: (o.A., 2019)

Wörterbuch (hash tables)

Wörterbücher (dictionaries) sind eine Sammlung von Schlüssel- und Wertpaaren. Vorteile:

- Kombination beliebiger Datentypen
- Effiziente Implementierung mithilfe von Hashtabellen

Key Wert

```
1  {} # leer
2  w = {1:'Wal', 2:'Hai'}
3  w[2]
4  w[5] = 'Ara'
5  w
6  list(w.keys())
7  [1,2,5]
8  'Wal' in w
9  2 in w
10 type(w) # <class 'dict'>
```

Einfügung

Source Code 20:
Wörterbuch-Operatoren

Funktionen für Wöterbücher	
`in`	Test auf Enthaltensein
`not in`	Nichtenthal-tensein
`d[k]`	Wert vom Key k
`d.values()`	Gibt alle Values aus
`d.keys()`	Gibt alle Keys aus

Table 11: Funktionen für Wörterbücher

Aufgabe D

Schreibe eine Funktion **newest**, die von den in einem Wörterbuch gespeicherten Studierenden den/die Studenten*in mit kleinster Matrikelnummer findet und ihn/sie als Tupel (Matrikelnummer, Name) ausgibt. Beispiel:

```
1 studierende = {1234567: 'Moritz', 7654321: 'Albert',
      1234321: 'Skadi', 5432657: 'Lukas'}
2 newest(studierende) # Ausgabe: (1234321, 'Skadi')
```

Source Code 21: Datensatz Studenten und Funktionsaufruf mit Ergebnis

Hinweis: Es kann die Funktion `sorted(liste)` benutzt werden, um die Matrikelnummern zu sortieren.
Achtung! Lösung:

```
1 def newest(studis):
2     lowestKey = sorted(studis.keys())[0]
3     return (lowestKey, studis[lowestKey])
```

Source Code 22: Lösung zur Aufgabe D

Inhalt

Fallunterscheidung


```
1  def signum42(n):
2      if n == 0:
3          return 0
4      elif n > 0:
5          if n == 42:
6              return 42
7          else:
8              return 1
9      else:
10         return -1
```

Source Code 23: Fallunterscheidung: if-then-else

Die **if-else-Anweisung** prüft ein Prädikat und führt zu einer Entscheidung, die den Programmablauf beeinflusst.

Mithilfe von **elif** kann zwischen mehr als nur 2 Fällen unterschieden werden.

Die **else-Anweisung** ist optional. Wird sie weg gelassen, wird am Ende der Fallunterscheidung fortgefahren.

Fallunterscheidungen können auch geschachtelt werden.

Inhalt

for-Schleife:

```
1  def sumAll(n):
2      sum = n
3      for i in range(n):
4          sum = sum+i
5          print(i)
6      return sum
```

Source Code 24: for-Schleife

for-Schleifen durchlaufen die Elemente der ihr übergebenen Liste. Der Zähler i wird in jedem Schleifendurchlauf verändert.
Besonderheit: Die Anzahl der Schleifendurchläufe steht zu Anfang fest.

while-Schleife:

```
1  def sumAll(n):
2      i, sum = 1, 0
3      while i <= n:
4          sum = sum+i
5          i = i+1
6      return sum
```

Source Code 25: while-Schleife

Solange die Bedingung erfüllt ist, führen **while-Schleifen** den Schleifenrumpf aus. Der Zähler `i` muss manuell verändert werden, damit die Schleife terminiert.
Besonderheit: Die Anzahl der Schleifendurchläufe braucht nicht festzustehen.

Aufgabe E

Schreibe eine Funktion **collatzList**, welche eine Zahl n bekommt und die Collatz-Folge von n bis 1 in Form einer Liste ausgibt. Beispiel:

$$\text{collatzList}(3) \mapsto [3, 10, 5, 16, 8, 4, 2, 1]$$

Dabei wird die n-te Collatz-Zahl c_n in Abhängigkeit ihres Vorgängers c_{n-1} wie folgt gebildet:

$$c_n = \begin{cases} 3 \cdot c_{n-1} + 1, & c_{n-1} \text{ ungerade} \\ c_{n-1}//2, & c_{n-1} \text{ gerade} \end{cases} \qquad \forall n \in \mathbb{N}$$

Achtung! Lösung:

```
1  def nextC(c):
2      if (c % 2 == 0):
3          return c//2
4      else:
5          return 3*c+1
```

```
1  def collatzList(n):
2      result = [n]
3      while (n > 1):
4          n = nextC(n)
5          result.append(c)
6      return result
```

Source Code 26: Lösung zur Aufg. E

Noch Fragen?

Python Tutorials: http://www.learnpython.org

Inhalt

Referenzen I

Epp, Dietrich (2012). *Python3.2 can not recognize UP/DOWN/LEFT/RIGHT keys in interpreter?*. Webseite. URL: https://stackoverflow.com/questions/10765441/python3-2-can-not-recognize-up-down-left-right-keys-in-interpreter (visited on 04/11/2019).

Esponda Agüero, M. (2012). "Objektorientierte Programmierung". Vorlesungsfolien.

Minimäxchen (2019). *Python*. Webseite. URL: https://wiki.ubuntuusers.de/Python (visited on 04/11/2019).

o.A. (2018). *Befehle*. Webseite. URL: https://www.shellbefehle.de/befehle (visited on 04/09/2019).

– (2019). *Python List*. Webseite. URL: https://www.programiz.com/python-programming/list (visited on 04/09/2019).

Oberneder, Armin (2018). *Cmd-Befehle unter Windows*. Webseite. URL: https://www.thomas-krenn.com/de/wiki/Cmd-Befehle_unter_Windows (visited on 04/09/2019).