

Einführungskurs Python

Dominik Dürschnabel

Mai 2019



Organisation und eine kurze Einführung in Python

Organisation und eine kurze Einführung in Python

Wer bin ich ?

Dominik Dürschnabel

- Wissenschaftlicher
Mitarbeiter am FG Wissensverarbeitung



Dominik Dürschnabel

- ▶ Wissenschaftlicher Mitarbeiter am FG Wissensverarbeitung
- ▶ Master Mathematik



Dominik Dürschnabel

- ▶ Wissenschaftlicher Mitarbeiter am FG Wissensverarbeitung
- ▶ Master Mathematik
- ▶ passionierter Programmierer (C/C++/Java/Python/etc. ...manchmal auch Clojure ;))



Dominik Dürschnabel

- ▶ Wissenschaftlicher Mitarbeiter am FG Wissensverarbeitung
- ▶ Master Mathematik
- ▶ passionierter Programmierer (C/C++/Java/Python/etc. ...manchmal auch Clojure ;))
- ▶ hat schon mal bei jemandem zugeschaut der Python programmiert



Organisation und eine kurze Einführung in Python

Warum ist Dominik hier?

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?
- ▶ weiß, dass Python in Wissenschaft wie Industrie sehr beliebt ist

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?
- ▶ weiß, dass Python in Wissenschaft wie Industrie sehr beliebt ist

Und woher kommt dieser Kurs?

- ▶ Fragment aus LLP-Kurs (**L**inux**L**aTeX**P**ython) 2014

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?
- ▶ weiß, dass Python in Wissenschaft wie Industrie sehr beliebt ist

Und woher kommt dieser Kurs?

- ▶ Fragment aus LLP-Kurs (**L**inux**L**aTeX**P**ython) 2014
- ▶ ...gehalten an der TU Dresden

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?
- ▶ weiß, dass Python in Wissenschaft wie Industrie sehr beliebt ist

Und woher kommt dieser Kurs?

- ▶ Fragment aus LLP-Kurs (**L**inux**L**aTeX**P**ython) 2014
- ▶ ...gehalten an der TU Dresden
- ▶ ...von Dr. Daniel Borchmann, Maximilian Marx und Tom Hanika

Dominik ...

- ▶ war der Erste der "Hier!"gerufen hat, als im FG gefragt wurde, wer den Python Kurs übernehmen möchte.
- ▶ (meint, Python ist eine gute Ergänzung zum Studium in Kassel)?
- ▶ weiß, dass Python in Wissenschaft wie Industrie sehr beliebt ist

Und woher kommt dieser Kurs?

- ▶ Fragment aus LLP-Kurs (**L**inux**L**aTeX**P**ython) 2014
- ▶ ...gehalten an der TU Dresden
- ▶ ...von Dr. Daniel Borchmann, Maximilian Marx und Tom Hanika
- ▶ ...von Tom Hanika hier an der Uni Kassel eingeführt

Organisation und eine kurze Einführung in Python

Warum seid Ihr hier? (Ab jetzt wird es interessant.)

Ihr ...

- ▶ seid nicht hier, weil Ihr Creditpoints erwartet.

Ihr ...

- ▶ seid nicht hier, weil Ihr Creditpoints erwartet.
- ▶ seid hier, um mehr, oder überhaupt etwas, über Python zu erfahren.

Ihr ...

- ▶ seid nicht hier, weil Ihr Creditpoints erwartet.
- ▶ seid hier, um mehr, oder überhaupt etwas, über Python zu erfahren.
- ▶ habt schon mal von Python gehört? (Jetzt nicken!)

Organisation und eine kurze Einführung in Python

Inhalt, Ablauf, Termine

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen
- ▶ Programmieren mit Python

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen
- ▶ Programmieren mit Python
- ▶ Wissenschaftliches Arbeiten mit Python (aka, tolle Libs)

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen
- ▶ Programmieren mit Python
- ▶ Wissenschaftliches Arbeiten mit Python (aka, tolle Libs)

Für Fragen, Kursmaterial und Anregungen:

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen
- ▶ Programmieren mit Python
- ▶ Wissenschaftliches Arbeiten mit Python (aka, tolle Libs)

Für Fragen, Kursmaterial und Anregungen:

- ▶ <https://www.uni-kassel.de/go/pythonkurs>

Inhalt, Ablauf, Termine

Termine:

6.Mai (Raum -1418) / 8.Mai (Raum -1418) / 13.Mai (Raum -1418)

Inhalt:

- ▶ Python-Grundlagen
- ▶ Programmieren mit Python
- ▶ Wissenschaftliches Arbeiten mit Python (aka, tolle Libs)

Für Fragen, Kursmaterial und Anregungen:

- ▶ <https://www.uni-kassel.de/go/pythonkurs>
- ▶ duerrschnabel@cs.uni-kassel.de

Organisation und eine kurze Einführung in Python

Fragen bis hierher?

Organisation und eine kurze Einführung in Python

Eine (sehr) kurze Geschichte von Python

Short history of Python (*...as in Monty Python!!*)

Short history of Python (*...as in Monty Python!!*)

- Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (*...as in Monty Python!!*)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint.
Features: *lambda*, *map*, *filter* and *reduce*



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (...as in Monty Python!!)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint. Features: *lambda*, *map*, *filter* and *reduce*
- ▶ Oktober 2000: Python 2.0 erscheint. DAS FEATURE: *List Comprehensions*



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (...as in Monty Python!!)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint. Features: *lambda*, *map*, *filter* and *reduce*
- ▶ Oktober 2000: Python 2.0 erscheint. DAS FEATURE: *List Comprehensions*
- ▶ Dezember 2008: Python 3.0 erscheint, ein **redesign** ⇒ Nicht Abwärtskompatibel!



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (*...as in Monty Python!!*)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint. Features: *lambda*, *map*, *filter* and *reduce*
- ▶ Oktober 2000: Python 2.0 erscheint. DAS FEATURE: *List Comprehensions*
- ▶ Dezember 2008: Python 3.0 erscheint, ein **redesign** ⇒ Nicht Abwärtskompatibel!
- ▶ Oktober 2018: Python 3.7.1



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (...as in Monty Python!!)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint. Features: *lambda*, *map*, *filter* and *reduce*
- ▶ Oktober 2000: Python 2.0 erscheint. DAS FEATURE: *List Comprehensions*
- ▶ Dezember 2008: Python 3.0 erscheint, ein **redesign** ⇒ Nicht Abwärtskompatibel!
- ▶ Oktober 2018: Python 3.7.1



Python 2 Support bis 2020



<http://www.flickr.com/people/>

52614599@N00

Short history of Python (...as in Monty Python!!)

- ▶ Februar 1991: GUIDO VAN ROSSUM veröffentlicht Python 0.9 auf alt.sources. (Zusammengebastelt aus *ABC* und *Modula-3*)
- ▶ Januar 1994: Python 1.0 erscheint. Features: *lambda*, *map*, *filter* and *reduce*
- ▶ Oktober 2000: Python 2.0 erscheint. DAS FEATURE: *List Comprehensions*
- ▶ Dezember 2008: Python 3.0 erscheint, ein **redesign** ⇒ Nicht Abwärtskompatibel!
- ▶ Oktober 2018: Python 3.7.1



Python 2 Support bis 2020

Wir lernen Python 3



<http://www.flickr.com/people/>

52614599@N00

Organisation und eine kurze Einführung in Python

Python in Buzzwords

Bingo anyone?

- ▶ Multiparadigmensprache?!:

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)
 - ▶ dynamisch (Laufzeit entscheidet ob Typ passt)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)
 - ▶ dynamisch (Laufzeit entscheidet ob Typ passt)
 - ▶ Duck-Typing (Wenn ich es addieren kann, ist es sowas wie ne Zahl)

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)
 - ▶ dynamisch (Laufzeit entscheidet ob Typ passt)
 - ▶ Duck-Typing (Wenn ich es addieren kann, ist es sowas wie ne Zahl)
- ▶ More buzzwords:

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)
 - ▶ dynamisch (Laufzeit entscheidet ob Typ passt)
 - ▶ Duck-Typing (Wenn ich es addieren kann, ist es sowas wie ne Zahl)
- ▶ More buzzwords:
 - ▶ Python wird *compiliert* zu Python-Bytecode

Bingo anyone?

- ▶ Multiparadigmensprache?!:
 - ▶ Imperativ (Anweisung nach Anweisung)
 - ▶ Strukturiert (Subroutine/Blöcke/etc)
 - ▶ Objekt-orientiert (wenn man will, sehr gut unterstützt)
 - ▶ Funktional („OK“: Kein automatisches Currying, tail recursion opt. ...)
- ▶ Typsystem:
 - ▶ stark (Es gibt Typen für Integer, Reals, Strings, usw.)
 - ▶ dynamisch (Laufzeit entscheidet ob Typ passt)
 - ▶ Duck-Typing (Wenn ich es addieren kann, ist es sowas wie ne Zahl)
- ▶ More buzzwords:
 - ▶ Python wird *compiliert* zu Python-Bytecode
 - ▶ Python ist (manchmal) langsam!?

Python-Grundlagen

Ziele des Abschnitts

Ziele

- ▶ „Python als Taschenrechner“
- ▶ Grundlagen: Syntax, Datentypen

Ziele des Abschnitts

Ziele

- ▶ „Python als Taschenrechner“
- ▶ Grundlagen: Syntax, Datentypen

(Ein bisschen) Interaktiv!

Ziele des Abschnitts

Ziele

- ▶ „Python als Taschenrechner“
- ▶ Grundlagen: Syntax, Datentypen

(Ein bisschen) Interaktiv!

Python installieren:

- ▶ <https://www.python.org/downloads/>
- ▶ hier: Python 3.6+

Hinweis für Einsteiger:

- ▶ Wir empfehlen für das Praktikum die Anaconda Distribution
- ▶ <https://www.anaconda.com/download/>
- ▶ Warum? Gutes Package-Management, viele relevante Pakete vorinstalliert, konzipiert für Data-Science.
- ▶ Allerdings: Nicht da hängen bleiben in der Zukunft ;)

Hallo.

```
1 print("Hello, \uworld!")
```

Hallo.

```
1 print("Hello, \uworld!")
```

Python 2:



```
1 from __future__ import print_function
```


Rechnen mit Python

- ▶ Arithmetik mit +, -, *, /, **, %
- ▶ Ganzzahlige Division mit //

Rechnen mit Python

- ▶ Arithmetik mit +, -, *, /, **, %
- ▶ Ganzzahlige Division mit //

```
1 >>> 23 + 42
2 65
3 >>> 3 / 2
4 1.5
5 >>> 3 // 2
6 1
7 >>> 42**2**2**2
8 93753749683698750476845056
```

Rechnen mit Python

- ▶ Arithmetik mit +, -, *, /, **, %
- ▶ Ganzzahlige Division mit //

```
1 >>> 23 + 42
2 65
3 >>> 3 / 2
4 1.5
5 >>> 3 // 2
6 1
7 >>> 42**2**2**2
8 93753749683698750476845056
```

Python 2:



```
1 from __future__ import division
```

Zahlentypen

- ▶ Ganzzahlen, beliebig genau
- ▶ Binäre Fließkommazahlen
- ▶ Komplexe Zahlen
- ▶ Rationale Zahlen (Modul `fractions`)
- ▶ Dezimale Fließkommazahlen (Modul `decimal`)

Zahlentypen

- ▶ Ganzzahlen, beliebig genau
- ▶ Binäre Fließkommazahlen
- ▶ Komplexe Zahlen
- ▶ Rationale Zahlen (Modul fractions)
- ▶ Dezimale Fließkommazahlen (Modul decimal)

```
1 >>> complex(0, 1)
2 1j
3 >>> 3 - 1j
4 (3-1j)
5 >>> (3-1j).conjugate()
6 (3+1j)
7 >>> (3 - 1j).real
8 3.0
9 >>> (3 - 1j).imag
10 -1.0
```

Zeichenketten

- ▶ In Python 3 alles Unicode! :)
- ▶ unveränderbar, jede Änderung erzeugt eine neue Zeichenkette

Zeichenketten

- ▶ In Python 3 alles Unicode! :)
- ▶ unveränderbar, jede Änderung erzeugt eine neue Zeichenkette

```
1 >>> "spam" 'eggs'
2 'spameggs'
3 >>> s = 'spam'
4 >>> s * 3 + 'eggs'
5 'spamspamspameggs'
6 >>> print('spam\neggs\nbacon')
7 spam
8 eggs
9 bacon
10 >>> print(r'spam\neggs\nbacon')  #RAW String
11 spam\neggs\nbacon
```

Listen

- ▶ veränderbare Folge von Elementen

Listen

- ▶ veränderbare Folge von Elementen

```
1 >>> l = list(range(1, 10))
2 >>> l
3 [1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> l[2]
5 3
6 >>> l[2:]
7 [3, 4, 5, 6, 7, 8, 9]
8 >>> l[:2]
9 [1, 2]
10 >>> l[::2]
11 [1, 3, 5, 7, 9]
12 >>> l[2:7:3]
13 [3, 6]
```

Mehr Listen

```
1 >>> l = list(range(1, 10))
2 >>> l
3 [1, 2, 3, 4, 5, 6, 7, 8, 9]
4 >>> l[2] = 23
5 >>> l
6 [1, 2, 23, 4, 5, 6, 7, 8, 9]
7 >>> list(reversed([1, 3, 2]))
8 [2, 3, 1]
9 >>> sorted([2, 7, 1, 8, 2, 8])
10 [1, 2, 2, 7, 8, 8]
11 >>> ", ".join(['eggs', 'bacon', 'spam'])
12 'eggs,bacon,spam'
```

Tupel

- ▶ unveränderbare Folge von Elementen

Tupel

- ▶ unveränderbare Folge von Elementen

```
1 >>> tom = (1,2,3)
2 >>> a, b, c = (1, 2, 3)
3 >>> a
4 1
5 >>> (b, c)
6 (2, 3)
7 >>> c, a, b = a, b, c
8 >>> (a, b, c)
9 (2, 3, 1)
10 >>> (1, 2, 3)[1]
11 2
```

Sets

- ▶ veränderbare Menge von unveränderbaren Elementen $\Rightarrow$?

Sets

- ▶ veränderbare Menge von unveränderbaren Elementen⇒?

```
1 >>> c = {'Mathe',5,True,2.812}
2 >>> c
3 {'Mathe', True, 2.812, 5}
4 >>> a = set(range(1, 10))
5 >>> b = set(range(3, 9))
6 >>> a
7 {1, 2, 3, 4, 5, 6, 7, 8, 9}
8 >>> a.union(b)
9 {1, 2, 3, 4, 5, 6, 7, 8, 9}
10 >>> a.issuperset(b)
11 True
12 >>> a.difference(b)
13 {1, 2, 9}
```

Dictionaries

- ▶ veränderbare Zuordnung von unveränderbaren Schlüsseln zu Werten

Dictionaries

- ▶ veränderbare Zuordnung von unveränderbaren Schlüsseln zu Werten

```
1 >>> d = {0: 'eggs', 1: 'bacon', 'spam': 'spam'}
2 >>> d
3 {0: 'eggs', 1: 'bacon', 'spam': 'spam'}
4 >>> d[1]
5 'bacon'
6 >>> d['spam']
7 'spam'
8 >>> d['eggs'] = 'spam'
9 >>> d
10 {0: 'eggs', 1: 'bacon', 'eggs': 'spam', 'spam': 'spam'}
```


Python-Grundlagen

Funktionen

Funktionen

- ▶ Einrückung als Syntaxelement
- ▶ Funktionen sind *first-class*

Funktionen

- ▶ Einrückung als Syntaxelement
- ▶ Funktionen sind *first-class*

```
1 >>> def f():
2     ...     "prints 'hello, world!'"
3     ...     print('hello, world!')
4     ...
5 >>> f()
6 'hello, world!'
7 >>> help(f)
8 Help on function f in module __main__:
9
10 f()
11     prints 'hello, world'
```

Mehr Funktionen

- ▶ Funktionen mit Argumenten
- ▶ Lambdas als anonyme Funktionen

Mehr Funktionen

- ▶ Funktionen mit Argumenten
- ▶ Lambdas als anonyme Funktionen

```
1 >>> def g(x, y):  
2     ...     return complex(y, x).conjugate()  
3     ...  
4 >>> g(23, 42)  
5 (42-23j)  
6 >>> g(y=23, x=42)  
7 (23-42j)  
8 >>> h = lambda x, y: complex(y, x).conjugate()  
9 >>> h(23, 42)  
10 (42-23j)
```

Noch mal ausführlich: Lambda / filter / reduce / map

```
1 >>> tolle_liste = [2, 24, 9, 23, 19, 18, 8, 6, 27]
2 >>> print(list(filter(lambda x: x % 3 == 0, tolle_liste)))
3 [24, 9, 18, 6, 27]
4 >>> print(list(map(lambda x: x**2, tolle_liste)))
5 [4, 576, 81, 529, 361, 324, 64, 36, 729]
6 >>> from functools import *
7 >>> print(reduce(lambda x,y: x+y,tolle_liste))
8 136
```

Python-Grundlagen

Kontrollfluss

Vergleiche

- ▶ Vergleiche mit <, <=, ==, !=, >=, >
- ▶ == vergleich Wert, nicht Referenz
- ▶ is vergleicht Referenz

Vergleiche

- ▶ Vergleiche mit <, <=, ==, !=, >=, >
- ▶ == vergleicht Wert, nicht Referenz
- ▶ is vergleicht Referenz

```
1 >>> 1 == 1
2 True
3 >>> 1 < 2 <= 3 != 4 >= 3 > 2
4 True
5 # alle paarweisen Vergleiche werden mit und verknüpft
6 >>> 1 < 2 <= 3 == 4 >= 3 > 2
7 False
8 >>> [1, 2, 3] == [1, 2, 3]
9 True
10 >>> [1, 2, 3] is [1, 2, 3]
11 False
```

Logische Ausdrücke

- ▶ and, or
- ▶ Kurzschlussauswertung

Logische Ausdrücke

- ▶ and, or
- ▶ Kurzschlussauswertung

```
1 >>> False and True
2 False
3 >>> False and 0 / 0 < 1
4 False
5 >>> 1 < 2 or False or 3 < 4
6 True
```

Bedingungen

```
1 def f(x):  
2     if -3 <= x <= 3:  
3         print(r'x\in(-3,3)')  
4     elif x < 0:    # muss es automatisch < -3 sein  
5         print('x<-3')  
6     else:  
7         print('x>3')
```

```
1 >>> f(0)  
2 x \in (-3, 3)  
3 >>> f(4)  
4 x > 3  
5 >>> f(-4)  
6 x < -3
```

Schleifen

- ▶ for und while

Schleifen

► for und while

```
1 >>> for i in range(1, 4):
2     ...     print(i)
3     ...
4     1
5     2
6     3
7 >>> i = 1
8 >>> while i < 4:
9     ...     print(i)
10    ...     i += 1
11    ...
12    1
13    2
14    3
```

Mehr Schleifen

```
1 ne_liste = ["Haus", 42, True]
2 for i in ne_liste:
3     print(i)
4     if i == "Haus":
5         print("Yay")
6     else:
7         print('Kein_Haus')
```

```
1 while ne_liste != [] :
2     print(ne_liste.pop())
```

```
1 noch_ne_liste = []
2 while len(noch_ne_liste) < 3 :
3     noch_ne_liste.append("ding")
4 print(noch_ne_liste)
```

Noch mehr Schleifen

- ▶ Iteration durch beliebige Sequenzen

Noch mehr Schleifen

- Iteration durch beliebige Sequenzen

```
1 >>> for c in 'spam':
2     ...     print(c)
3     ...
4 s
5 p
6 a
7 m
8 >>> for x in {'eggs', 23, 'bacon', 'spam'}:
9     ...     print(x)
10    ...
11 eggs
12 spam
13 bacon
14 23
```

Fragen soweit?