Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozentin: Esther Seyffarth

1. Foliensatz

Organisatorisches

- Dozentin: Wiebke Petersen
  petersen@phil.uni-duesseldorf.de
  Sprechstunde: Mo. 17 Uhr
- Co-Dozentin: Esther Seyffarth
  esther.seyffarth@uni-duesseldorf.de
- Vorsicht: Die Sitzungen finden in wechselnden Räumen statt.
- Alle wichtigen Informationen zur Organisation des Kurses finden Sie auf der Kurswebsite:
  http://user.phil-fak.uni-duesseldorf.de/~petersen/SoSe15_Python/SoSe15_Python_Petersen.html

Vorkenntnisse

- Es sind keinerlei Vorkenntnisse erforderlich!
- Sollten Sie bereits mehrere Programmiersprachen kennen und/oder häufiger eigene Programme schreiben, dann ist dieser Kurs nicht für Sie geeignet.
  Sie können sich Python effizienter selbst beibringen.
- Über welche Vorkenntnisse verfügen Sie?
- Warum nehmen Sie an dieser Veranstaltung teil?

Scheinerwerb

Beteiligungsnachweise

- Einige Aufgaben, die zumeist während der Sitzungen bearbeitet werden.
- Programmier- und Theorieaufgaben in der letzten Sitzung
- Zur Vorbereitung auf die Aufgabe sollten Sie ein persönliches Python reference sheet erstellen mit
  - allen Informationen zu den eingeführten Befehlen und Konzepten, die Sie für das Entwickeln eigener Programme benötigen.
  - pro Sitzungswoche max. 1 Seite
  - die reference sheets müssen am 26.5. und am 7.7. eingereicht werden (elektronisch). Für die Programmieraufgabe dürfen nur die zuvor eingereichten Seiten genutzt werden.

Ziele

In diesem Kurs lernen Sie

- ein Problem algorithmisch zu behandeln
- einfache Programme in Python zu schreiben
- die Benutzung von NLTK

Was ist Programmieren?

- Programmieren ist ein Handwerk,
  dessen Produkt ist ein Programm bzw. Quellcode
- Programme sollen einen Zweck erfüllen:
  - Beispiel: Segmentiere Text in Worte (Tokenisierung)
- Idee: Ich kann das mit Word machen! (Suchen und ersetzen)
- Probleme:
  - Textmenge (Korpus) ist zu groß
  - Word ist Text- und keine Datenverarbeitung!
  - sehr wahrscheinlich kein spezielles, angepasstes Verfahren
  - fehleranfällig weil intransparent, langsam ...
- Idee 2: Einen Algorithmus implementieren, der Wortgrenzen findet
- Algorithmen sind Handlungsanweisungen (Klassisches Beispiel: Kochrezepte) mit dem Ziel der Problemlösung. Sie sind ein Teil des Programms.

Warum (in Python) programmieren?

- Wiederverwendbar: Automatisierte Lösungen
- Synergieeffekte: Programme können sich ergänzen
- Spezialisierung: Ich schaffe meinen eigenen Werkzeugkasten
- Transparenz: Quellcode ist auch später les- und nachvollziehbar
- Flexibilität: Fremder Python-Quellcode ist fast immer nutz- und anpassbar, weil frei lizensiert.
- Wir müssen das Rad häufig nicht neu erfinden: Es existieren viele große Projekte und Bibliotheken, etwa das Natural Language Processing Toolkit (NLTK)

Python – Geschichte und Eigenschaften

- Ende der 1980er Jahre von Guido van Rossum entwickelt
- Ziel: Programmieren soll einfach sein und Spaß machen. Dazu setzt die Sprache auf eine besonders übersichtliche und lesbare Syntax
- Unter Linux und Mac OS X ist Python in der Regel vorinstalliert
- Derzeit aktuell: Python 3.4.
- Einsatzgebiete
  - Sehr gut geeignet, um Ideen schnell zu testen (Rapid Prototyping)
  - Sehr gut für wissenschaftliche Aufgaben geeignet, viele Bibliotheken und Module
- Gute Dokumentation und viel Literatur vorhanden

Zen of Python

Das Zen of Python (`http://www.python.org/dev/peps/pep-0020/`) besteht aus 20 Grundsätzen, die den Charakter von Python bestimmen. Hier eine Auswahl:

- Beautiful is better than ugly.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- Special cases aren't special enough to break the rules.
- There should be one– and preferably only one –obvious way to do it.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

(Vor-)Lesbarkeit

```
1  import random
2  even = 0
3  odd = 0
4  for counter in range(0,10):
5    number = random.randint(0,100)
6    if number % 2 == 0:
7      print(number, "is even")
8      even = even + 1
9    else:
0      print(number, "is odd")
1      odd = odd + 1
2  print("Found", even, "even and", odd, "odd numbers")
```

- Die Einrückungen geben eine klare Struktur und sind Teil der Syntax
- kaum Klammern, keine geschweiften Klammern
- klare Benennung der Funktionen und Bedingungen sowie der Schleife
- Variablen tragen passende Namen
- **Wichtig:** Gewöhnen Sie sich eine sprechende und konsistente Benennung an! Für Variablen und Funktionen Kleinbuchstaben. Bei Variablen Typ als **Suffix**: `words_list`. Bei Funktionen geeignetes Präfix: `get_words()`
- Operatoren verhalten sich erwartungsgemäß

Arbeitsablauf

- Interaktiv Programmieren mittels des **Interpreters**. Oder:
- Quellcode mittels einer **IDE** oder geeignetem Editor erstellen (mindestens Syntaxhighlighting, also nicht Word!!)
- durch einen **Compiler** in ein Programm übersetzen
- Programm mittels der Eingabeaufforderung oder **Python-Shell** ausführen

- Keine Panik: Fettgedruckte Begriffe werden detailliert erklärt oder es gibt Beispiele

Compiler

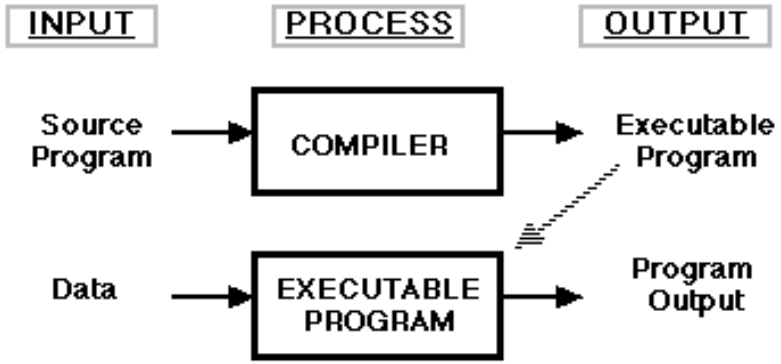
"Ein Compiler (auch Übersetzer oder Kompilierer genannt) ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm – genannt Quellprogramm – in ein semantisch äquivalentes Programm einer Zielsprache (Zielprogramm) umwandelt.
Üblicherweise handelt es sich dabei um die Übersetzung eines von einem Programmierer in einer Programmiersprache geschriebenen Quelltextes in Assemblersprache, Bytecode oder Maschinensprache."
(Aus `http://de.wikipedia.org/wiki/Compiler`)

Interpreter

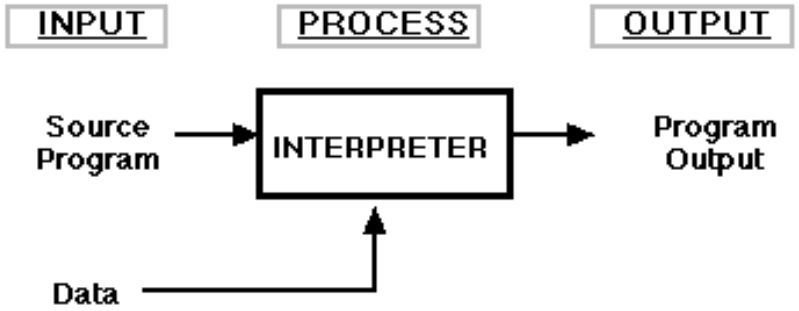
"Ein Interpreter (im Sinne der Softwaretechnik) ist ein Computerprogramm, das einen Programm-Quellcode im Gegensatz zu Assemblern oder Compilern nicht in eine auf dem System direkt ausführbare Datei umwandelt, sondern den Quellcode einliest, analysiert und ausführt. Die Analyse des Quellcodes erfolgt also zur Laufzeit des Programms."
(Aus `http://de.wikipedia.org/wiki/Interpreter`)

Zusammenfassung: Compiler und Interpreter

- Compiler: Erstellt aus Quelltext ein Zielprogramm



- Interpreter: Führt Eingaben direkt aus:

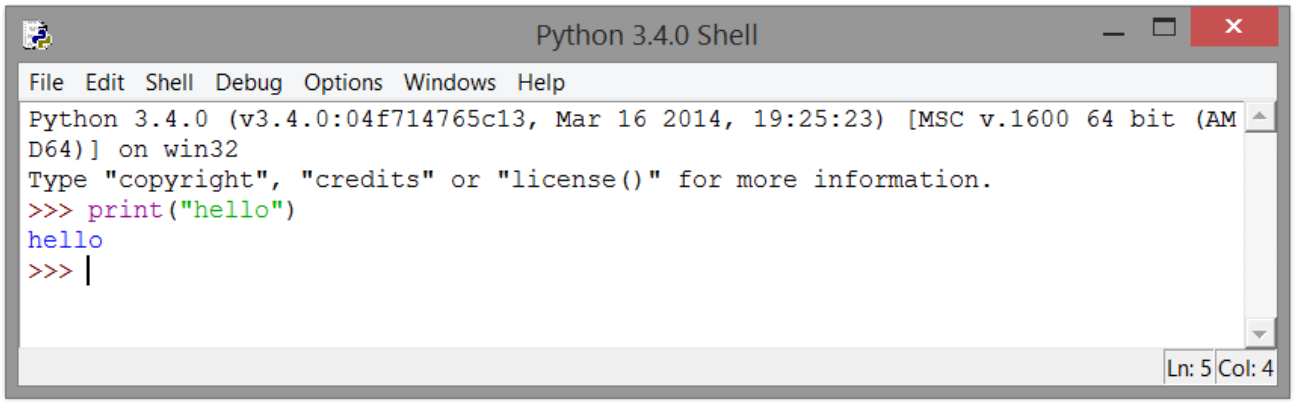


- Python steht zwischen reinen Interpreter- und reinen Compiler-Sprachen: Programme werden zur Laufzeit in Bytecode übersetzt, der dann vom Python-Interpreter ausgeführt wird

Quelle Grafiken: `http://www.cs.uaf.edu/~cs631/notes/node2.html` (Stand: 17.03.2015)

Materialien: Was brauche ich?

- Damit Python benutzt werden kann, müssen ein Interpreter und ein Compiler installiert sein. Der Interpreter läuft als Programm in einer Shell. Beim Installieren des aktuellen Python-Pakets werden alle benötigten Programmelemente mit eingerichtet.



Python installieren (Übersicht)

- **WICHTIG**: Verwenden Sie keine Leerzeichen, Umlaute oder Sonderzeichen in Ordner- oder Dateinamen! Python sollte also nicht in C:/Dokumente und Einstellungen/ o.ä. installiert werden, sondern beispielsweise direkt ins Laufwerk C:/.
- Download und Installation von Python werden im Kurs besprochen. Falls später beim Installieren zuhause Probleme auftauchen: Eine Anleitung zum Einrichten von Python findet sich unter `https://wiki.python.org/moin/BeginnersGuide/Download`.
- In diesem Kurs wird mit der Python-Version **3.4.3** gearbeitet.
- Auf den Rechnern in der Uni läuft Windows. Python kann aber auch unter MacOS und Linux verwendet werden, Anleitungen hierzu finden sich ebenfalls unter der obenstehenden URL.

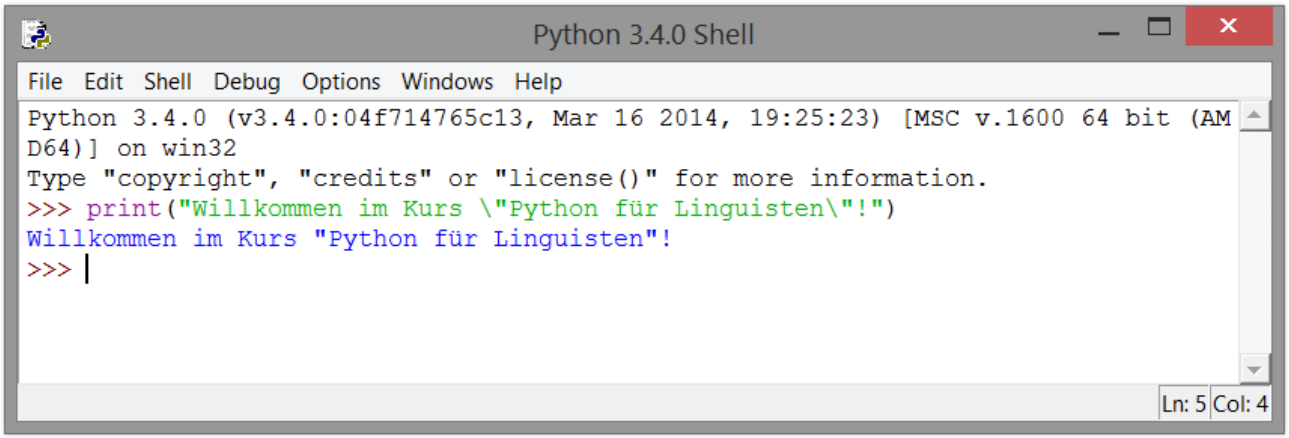
Python installieren (Windows)

- Download-Link für Rechner mit 64bit-Architektur (um die Architekturart zu erfahren: Rechtsklick auf "Computer", Kategorie "Systemtyp"):
  `https://www.python.org/ftp/python/3.4.3/python-3.4.3.amd64.msi`
- Download-Link für Rechner mit 32bit-Architektur:
  `https://www.python.org/ftp/python/3.4.3/python-3.4.3.msi`
- Zum Installieren sind administrative Berechtigungen nötig.
- Installer (.msi) ausführen und den Anweisungen folgen
- Lizenzbedingungen zustimmen
- Die Python-Shell heißt **IDLE**. Sie kann im Startmenü durch das Tippen von "IDLE" aufgerufen werden oder durch Starten der Datei C:/Python34/Lib/idlelib/idle.pyw.

Python-Pfad einrichten

- Damit Windows weiß, wie Python-Programme ausgeführt werden sollen, muss gegebenenfalls noch die Umgebungsvariable PATH angepasst werden.
- Um zu überprüfen, ob der Python-Pfad korrekt gesetzt ist, kann man in der Windows-Eingabeaufforderung "python" eintippen.
- Falls ein Fehler ausgegeben wird:
  - Rechtsklick auf "Computer" → Erweiterte Systemeinstellungen → Umgebungsvariablen
  - Im unteren Teil des Fensters die Variable "Path" oder "PATH" suchen und bearbeiten. Der anzugebende Pfad ist das Installationsverzeichnis von Python, beispielsweise C:/Python34/. Einzelne Einträge in der Path-Variable werden mit ; getrennt.

IDLE Python Gui



IDLE Python Gui: Editor

- IDLE enthält zwei Komponenten: den Editor und die interaktive Shell. Letztere wird zuerst gestartet (siehe vorherige Abbildung).
- Der Editor kann über `File` → `New File` gestartet werden.
- Im Editor können größere Programme geschrieben und gespeichert werden. Per `F5` werden sie ausgeführt.
- Wenn Fehler auftreten, wird immer eine Zeilennummer angegeben, in der der Fehler gefunden wurde. Die Zeilennummer der aktuellen Cursorposition wird in der unteren rechten Ecke der Shell angezeigt.
- Die Dateien sollten systematische Namen haben, etwa `p1_integer.py` für ein Skript aus der ersten Sitzung, in dem es um den Umgang mit Integern geht. Verwenden Sie Kleinbuchstaben und keine Leerzeichen, Umlaute oder Sonderzeichen!
- In IDLE sind einige nützliche Tastaturkürzel eingebaut, die in den Menüleisteneinträgen angezeigt werden. Sie erleichtern die Arbeit erheblich.
- Python-Befehle können während der Eingabe per `TAB` vervollständigt werden.

IDLE Python Gui: Shell

- Python-Befehle, die in der interaktiven Shell eingegeben werden, werden direkt nach dem Bestätigen der Eingaben mit `Enter` vom Interpreter ausgeführt.
- Per `ALT + P` und `ALT + N` kann man im Befehlsverlauf zurück- und vorblättern.
- Die eingegebenen Befehle und ihre Ausgaben können mit `Strg + S` oder durch `File` → `Save` gespeichert werden.
- Die Shell kann durch `Strg + F6` oder durch `Shell` → `Restart Shell` neu gestartet werden.
- Die Bedeutung der einzelnen Farben des Syntaxhighlighting kann man sich unter `Options` → `Configure IDLE` → `Highlighting` ansehen.

Ausprobieren von Python

- Wir beginnen zunächst mit der Eingabe einzelner Befehle im Interpreter.
- Der Prompt ">>>" zeigt an, dass der Interpreter auf eine Eingabe wartet.
- Beispiele:
  - `1 + 2` führt zur Ausgabe `3`
  - `10 / 2` führt zur Ausgabe `5.0`
  - `round(5.5)` führt zur Ausgabe `6`
  - `x = 5` und `y = 6` und `min(x,y)` führt zur Ausgabe `5`
  - `sprache = "Python"` und `attribut = "is great!"` führt mit `sprache + attribut` zu `...`

Variablen

- Werte wie Zahlen oder Zeichenketten können in Variablen gespeichert werden. Der eigentliche Inhalt der Variable ist dabei der Verweis auf die Speicherzelle im Arbeitsspeicher, in der der Wert gespeichert ist.
- Jede Variable hat einen Datentyp. Python ist im Gegensatz zu z.B. Java eine dynamisch typisierte Sprache.
- Variablen können zum Beispiel:
  - ausgegeben (`print(var)`),
  - verändert (`a = a + 1` bzw. `int("1")`) oder
  - kopiert (`a = b`) werden.

Grundlegende Datentypen

- Die Ausgabe des Befehls `type(variable)` ist der Typ von `variable`.
- Welchen Typ haben die folgenden Objekte?
  - Ganzzahlen: `0,1,2`
  - Gleitkommazahlen: `3.1415926535897931`
  - Text: `"Python"`
  - `False` und `None` (einfach mal ausprobieren)

Grundlegende Datentypen

- `int` für Ganzzahlen: `0,1,2`
- `float` für Gleitkommazahlen: `3.1415926535897931`
- `str` für `"Text"`
- "Besondere" Datentypen sind:
  - `bool` für `True` oder `False`
  - `None` (noch) kein Wert zugewiesen
- Es gibt auch komplexere Datentypen, und man kann in Python auch eigene Objektklassen definieren...
- ... es ist aber auch möglich, ohne Objektorientierung in Python gute und mächtige Programme zu schreiben.
- Wir werden in den nächsten Wochen noch weitere nützliche Datentypen kennenlernen.

Übung Operatoren

- Experimentieren sie mit einfachen Operatoren wie `+`, `-`, `*`, `/` und `//`.
- Was machen diese Operatoren?
- Wo müssen sie syntaktisch in einem Ausdruck stehen?
- Welche Typen fordern sie?

Übung Operatoren (II)

- Schreiben sie einen Pythonausdruck, der zwei Variablen `x` und `y` verwendet und der zum Beispiel für `x=3` und `y=7` ausgewertet wird zu

  `'3*7=3+3+3+3+3+3+3=21'`

- Versuchen Sie sich zunächst an einem einfacheren Ausdruck, zum Beispiel dem der folgendes produziert:

  `'3*7=21'`

Schreiben von Python-Programmen

- Um Programme erneut verwenden zu können, müssen die Anweisungen in eine Datei geschrieben werden, da Python interaktive Eingaben nach Beenden des Interpreters verwirft.
- Hierfür eignet sich der Editor von IDLE oder ein anderer Editor mit Syntaxhighlighting, wie beispielsweise Notepad++.
- Speichern mit Endung "`.py`" (Beispiel "`script.py`")
- Ausführen von der Kommandozeile mit "`python script.py`"
- Alternativ: Programme aus dem Editor von IDLE werden mit ‘run module’ geladen.
- Kurze Anmerkungen oder Anweisungen, die nicht ausgeführt werden sollen (z.B. zum Debuggen), werden mit # am Anfang der Zeile als Kommentar markiert.

“Hello world!”

```
1 # Print "Hello world!" to the screen
2 print("Hello world!")
```

- Kommentarzeichen: `#` (alles, was auf `#` folgt, wird nicht als zum Programm gehörig interpretiert)
- Kommentare sind wichtig, sie helfen den Code zu verstehen
- Gewöhnen Sie sich an, sorgfältig zu kommentieren!
- `print` ist eine Anweisung
- `Hello world!` ist das Argument von `print`
- Jede Anweisung bekommt eine eigene Zeile!
- In Python gibt es, im Gegensatz zu Sprachen wie Perl oder Prolog, kein besonderes Zeichen zum Zeilenende

Textausgabe

```
1 print("Hello world!")
2 print("Hello " + " world!")
3 print("Hello ", " world!")
4 print("Hello {}!".format("world"))
5 print("{} plus {} ist {}.".format(3, 6, 3+6))
```

Zeile 1 Die einfachste Form der Textausgabe: gibt eine fest definierte Zeichenkette aus

Zeile 2 Wie Zeile 1, allerdings mit Stringkonkatenation

Zeile 3 mehrstellige Textausgabe: gibt ein Leerzeichen zwischen den Argumenten aus.

Zeile 4 Textausgabe mit Formatstring: gibt eine Zeichenkette mit einem Platzhalter `{}` aus. Der Platzhalter wird in der Ausgabe durch den Wert `"world"` ersetzt.

Zeile 5 Ausgabe mit mehreren Platzhaltern: die Platzhalter `{}` werden entsprechend ihrer Reihenfolge ersetzt

Texteingabe

```
1 eingabe = input()
2 print(eingabe)
```

- Die Funktion `input()` liest eine Eingabe von der Standardeingabe (Tastatur).
- In Zeile 1 wird die Benutzereingabe in der Variable `eingabe` gespeichert.

```
1 name = input("What is your name? ")
2 print("Hello", name)
```

- `input()` wird hier mit dem zusätzlichen Argument `"What is your name? "` aufgerufen. Die Funktion gibt so zunächst das Argument auf dem Bildschirm aus und wartet dann auf eine Benutzereingabe.

Übungseinheit

- Schreiben Sie ein Programm, das Sie, wenn es gestartet wird, zunächst nach dem Vornamen und dann nach dem Nachnamen fragt und sie anschließend begrüßt.
- Schließen Sie die IDLE.
- Führen Sie ihr Programm aus.
- Beispiel-Output: `"Guten Tag, Wiebke Petersen!"`

Übungseinheit, Teil 2

- Erweitern Sie das vorige Programm so, dass Sie nicht mit vollem Namen begrüßt werden, sondern nur mit Ihren Initialien.
- Beispiel-Output: `"Hallo, W.P.!"`

Nützliche Funktionen von IDLE

- Durch vorher eingegebene Befehle blättern: `ALT+P` (zurück), `ALT+N` (vor)
  - Sie können auch blättern, wenn Sie bereits angefangen haben, etwas einzugeben. IDLE filtert die vorherigen Eingaben dann nach dem Beginn der aktuellen Zeile.
  - So können Sie z.B. `>>> x=` eingeben und durch alle bisherigen Zuweisungen von x blättern.
- Bei der Eingabe von Keywords (Python-spezifische Wörter, die eine eigene Funktionalität haben und von IDLE in einer anderen Farbe angezeigt werden als normaler Text) oder Variablen, die Sie bereits definiert haben, können Sie sich Tipparbeit sparen, indem Sie `TAB` drücken, nachdem Sie den Anfang des Wortes geschrieben haben.
  - Aus `>>> pri` wird `>>> print`.
  - Wenn die Eingabe noch nicht eindeutig ist, erscheint ein Menü, aus dem Sie das richtige Keyword aussuchen können.

Materialien

- **Tutorial:**
  - `http://www.python-kurs.eu/python3_kurs.php`
  - `https://docs.python.org/3.4/tutorial/`
  - **Achtung:** Es gibt viele interaktive Tutorials zu Python im Internet, die auf Python 2.7 basieren. Wenn Sie die Tutorials nutzen, achten Sie auf die Unterschiede zwischen den Pythonversionen!
- **The Hitchhiker's Guide to Python**
  `http://docs.python-guide.org/en/latest/`
- **Dokumentation** zu Python 3.4.3:
  `https://docs.python.org/3.4/`
- **Bei unerwarteten Programmierproblemen:**
  `https://stackoverflow.com`
- **Visualizer**: `http://pythontutor.com/visualize.html`
  - Im Visualizer können Sie Python 3.3 auswählen. Im Rahmen dieses Kurses sind die Unterschiede zwischen Python 3.3 und Python 3.4 vernachlässigbar.

Literatur

- Allen B. Downey (2012). Think Python: How to Think Like a Computer Scientist. O'Reilly Media (online: `http://www.greenteapress.com/thinkpython/`)
- Mark Pilgrim (2004). Dive into Python: Python from novice to pro. Online book: `http://www.diveintopython.net/`
- Steven Bird, Ewan Klein, and Edward Loper (2009). Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit. O'Reilly Media (online: `http://nltk.org/book/`)
- Michael Dawson (2010): Python Programming for the absolute beginner. 3rd edition. Course Technology / Cengage Learning
- Jacob Perkins (2010): Python Text Processing with NLTK 2.0 Cookbook. Packt Publishing.