Python für Linguisten

Dozentin: Wiebke Petersen & Co-Dozent: Valentin Heinz

1. Foliensatz

Organisatorisches

- Dozentin: Wiebke Petersen
  petersen@phil.uni-duesseldorf.de
  Sprechstunde: Mo. 16:30-17:30
- Co-Dozent: Valentin Heinz
  heinzv@phil-fak.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/SoSe13_Python/SoSe13_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

- Programmieraufgabe 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 Sitzung max. 1 Seite
  - die reference sheets müssen am 27.5. und am 8.7. 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: Du schaffst deinen eigenen Werkzeugkasten
- Transparenz: Quellcode ist auch später les- und nachvollziehbar
- Flexibilität: Fremder Python-Quellcode ist fast immer nutz- und anpassbar, weil frei lizensiert.
- Du musst 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 3.3. Wir benutzen allerdings 2.7, da manche große Pakete (etwa matplotlib oder nltk) noch nicht portiert sind. 2.7 ist die aktuellste Version vor 3.0
- Allerdings werden wir so viele Funktionen aus Python 3 benutzen wie möglich, die wir (etwa für `print()` mit `from __future__ import print_function`) importieren
- 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  # print ist eine Funktion in Python 3 geworden,
2  # die wir in 2.7 benutzen wollen:
3  from __future__ import print_function
4  import random
5  even = 0
6  odd = 0
7  for counter in range(0,10):
8    number = random.randint(0,100)
9    if ((number % 2) == 0):
0      print(number,"is even")
1      even = even + 1
2    else:
3      print(number,"is odd")
4      odd = odd + 1
5  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:** Gewöhne dir 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

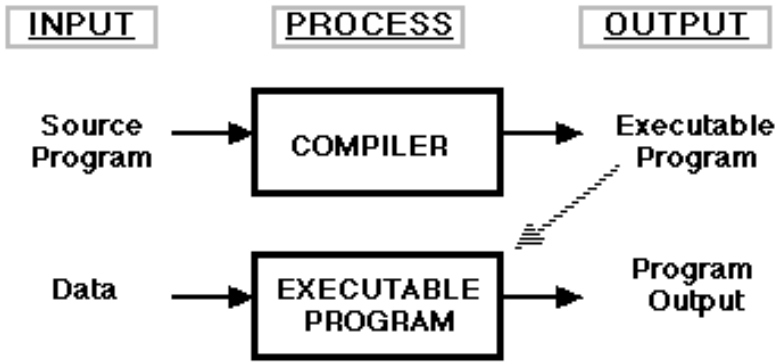
"Ein Compiler (auch Übersetzer oder Kompilierer genannt) ist ein Computerprogramm, das ein in einer Quellsprache geschriebenes Programm – genannt Quellprogramm – in ein semantisch äquivalentes Programm einer Zielsprache (Zielprogramm) umwandelt.
Üblicherweise handelt es sich dabei um die Übersetzung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

Interpreter

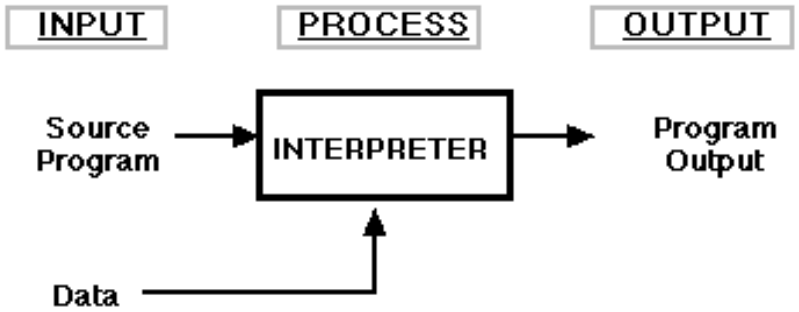
"Ein Interpreter (im Sinne der Softwaretechnik) ist ein Computerprogramm, das einen Programm-Quellcode im Gegensatz zu Assemblern oder Compilern nicht in eine auf dem System direkt ausführbare Datei umwandelt, sondern den Quellcode einliest, analysiert und ausführt. Die Analyse des Quellcodes erfolgt 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:

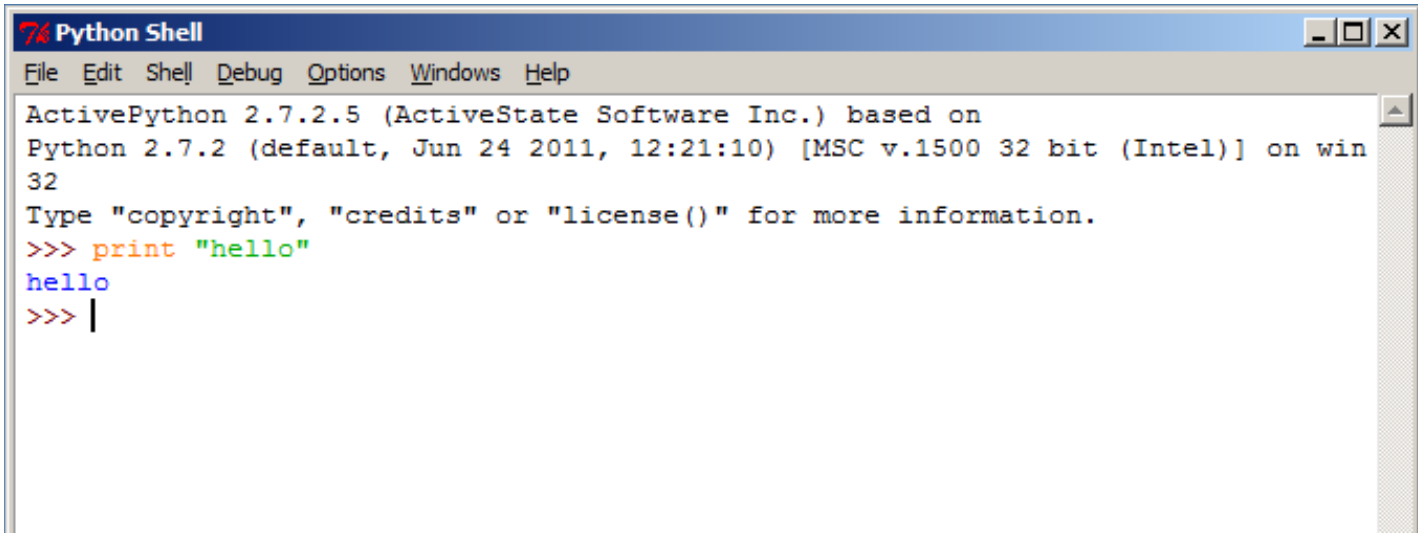


- Python steht zwischen reinen Interpreter- und reinen Compiler-Sprachen: Programme werden zur Laufzeit in Bytecode übersetzt, der dann vom Python-Interpreter ausgeführt wird

Quelle Grafiken: `http://www.cs.uaf.edu/~cs631/notes/node2.html` (Stand: 05.04.2013)

Materialien: Was brauche ich?

Du brauchst einen Python-Interpreter und einen Compiler. Der Interpreter läuft als Programm in einer Shell. Active Python ist (in der Community-Edition) Freeware für Windows und stellt vorkompiliert eine Shell mit Interpreter, einen Compiler, sowie einen Editor bereit.



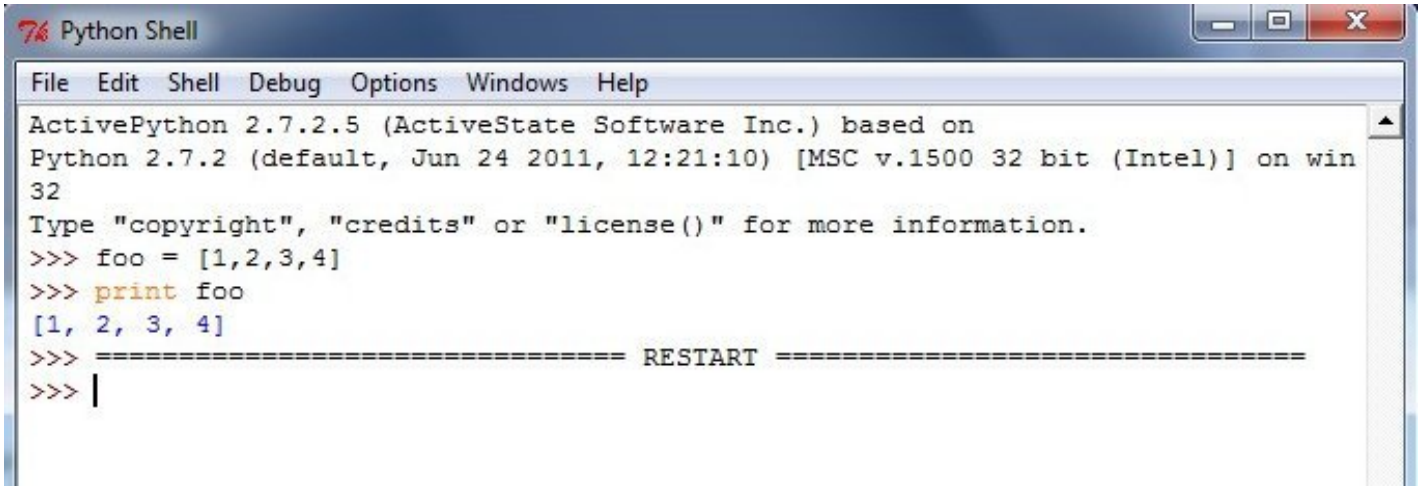
ActivePython installieren (Übersicht)

- **WICHTIG:** Verwende keine Leerzeichen in Ordner- oder Dateinamen!!
- **Konvention:** Tastaturkürzel werden in Großbuchstaben angegeben: `STRG + S` ist ungleich `STRG + SHIFT + S`
- für Ungeduldige und **bei Problemen**: Download und Installation von ActivePython sind hier dokumentiert: `http://docs.activestate.com/activepython/2.7/installnotes.html`. Falls du die Dokumentation direkt liest, denk daran: **Wir benutzen Python 2.7, du brauchst also derzeit Version 2.7.2.5.**
- MacOS bzw. Linuxnutzer sollten keine Probleme haben, wenn sie ihren Editor und ein Terminal starten können. Falls du dich doch für ActivePython entscheidest, besuch obigen Link und folge den detaillierten Anweisungen zu deinem Betriebssystem.

ActivePython installieren (Windows)

1. wenn du weisst, dass du eine 64bit-Architektur hast, lade `http://downloads.activestate.com/ActivePython/releases/2.7.2.5/ActivePython-2.7.2.5-win64-x64.msi` herunter
2. falls nicht, oder du dir nicht sicher bist, lade `http://downloads.activestate.com/ActivePython/releases/2.7.2.5/ActivePython-2.7.2.5-win32-x86.msi` herunter
3. stelle sicher, dass du administrative Berechtigungen hast
4. doppelklicke den Installer (.msi) und folge den Anweisungen
5. stimme den Lizenzbedingungen zu
6. installiere alle Features: `pywin32`, `doc` und `register` (default)
7. über das Startmenü solltest du ActivePython nun wie gewohnt erreichen. **Starte nun IDLE Python Gui**

IDLE Python Gui



IDLE Python Gui: Editor

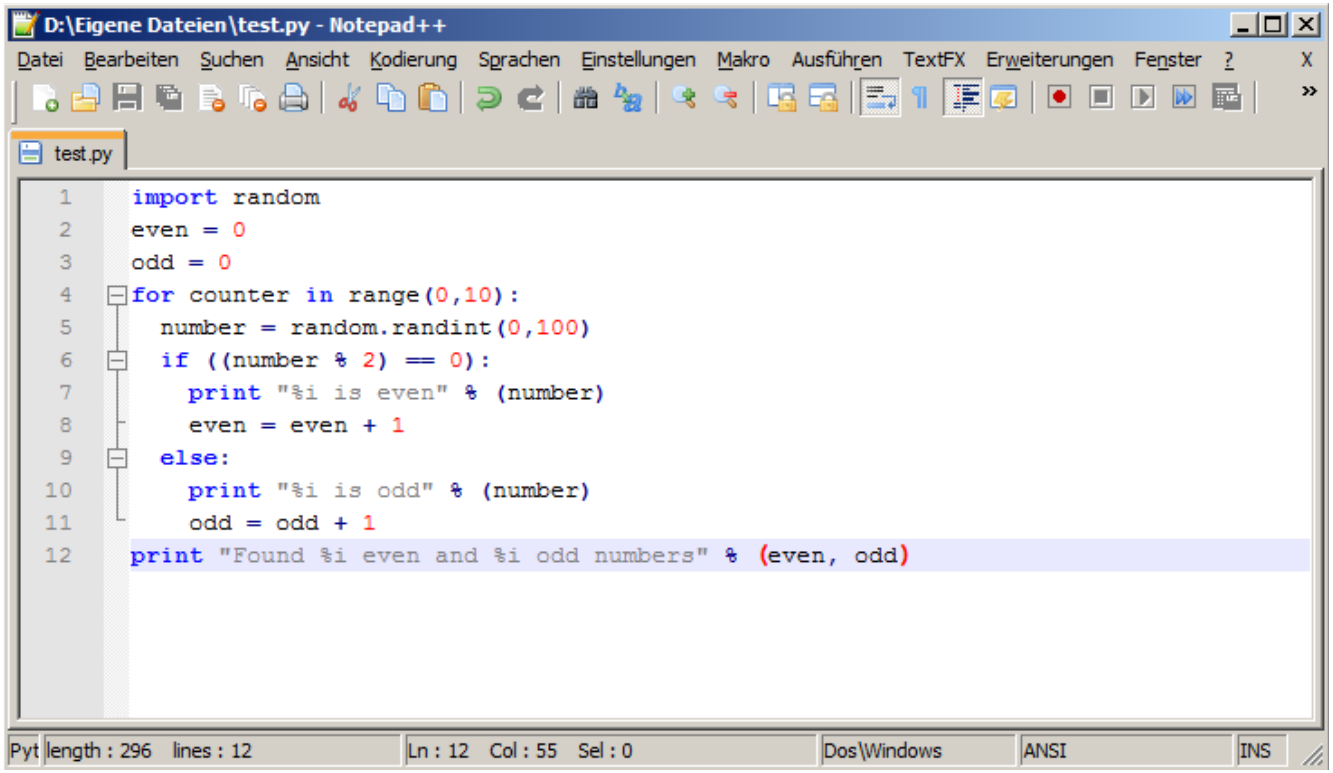
- die Anwendung besteht aus zwei Fenstern: dem Editor und der Shell. Letztere wird zuerst gestartet (siehe vorherige Abbildung)
- den Editor kannst du über `File` → `New Window` starten
- hier solltest du größere Programme schreiben und per `F5` ausführen, du musst jedoch zuvor speichern (automatische Aufforderung)
- **WICHTIG:** benenne deine Dateien systematisch, etwa `p1_integer.py` für ein Skript aus der ersten Sitzung, in welchem du mit Integern gearbeitet hast. Verwende Kleinbuchstaben und keine Leerzeichen!
- einen Überblick über die Tastaturkürzel erhältst du durch die einzelnen Reiter, erstelle dir einen Merkzettel
- du kannst Befehle per `TAB` vervollständigen
- die Ausführung findet durch den Interpreter im Rahmen der Shell statt

IDLE Python Gui: Shell

- diese Shell ist **NICHT** die Python Interactive Shell, die im Startmenü aufgeführt ist!
- diese Shell ist das erste Fenster, das sich öffnet, wenn du Python IDLE Gui aufrufst.
- in der Shell kannst du Befehle eingeben, ausführen und per `STRG + P` und `STRG + N` im Befehlsverlauf "blättern"
- Befehle können mit `Strg + S` gespeichert werden, oder durch `File` → `Save`
- die Shell kann durch `Strg + F6` neu gestartet werden, dies kann auch durch den Reiter `Shell` → `Restart Shell` erfolgen
- unter dem Reiter `Edit` findest du hilfreiche Funktionalitäten, wie etwa das Zurücknehmen von Änderungen
- um das Syntaxhighlighting zu verstehen, kannst du unter `Options` → `Configure IDLE` (neues Fenster öffnet sich) → `Highlighting` eine Vorschau aufrufen

Materialien: Editor

Notepad++ ist ein übersichtlicher und freier Editor für Windows, mit vielen Features wie Syntax-Highlighting und Auto-Completion.
Benutze niemals ein Textverarbeitungsprogramm wie Word!!
`http://sourceforge.net/projects/notepad-plus/files/`



Ausprobieren von Python

- Man muss nicht gleich ein ganzes Python-Programm schreiben
- Kleine Python-Anweisungen lassen sich direkt in den Python-Interpreter eingeben
- Dazu in einer Kommandozeile das Programm "`python`" starten
- Das Prompt ">>> " zeigt an, dass eine Anweisung eingegeben werden kann
- Beispiele:
  - `1 + 2` führt zur Ausgabe "3"
  - `10 / 2` führt zur Ausgabe 5
  - `round(5.5)` führt zur Ausgabe "6.0"
  - `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

- Variablen sind Container für Inhalte
- diese haben einen Datentyp, dazu gleich mehr
- Variablen können zum Beispiel:
  - ausgegeben (`print var`),
  - verändert (`a = a + 1` bzw. `int('1')`), oder
  - kopiert (`a = b`) oder
  - gelöscht (`del a b var`) werden
- Mit Ganzzahlen und Gleitkommazahlen können mathematische Operationen ausgeführt werden. Was fällt dir auf?

Grundlegende Datentypen

- Variablen haben einen Typ, den du mit `type(variable)` herausfinden kannst
- Finde folgende Typen:
  - Ganzzahlen: `0,1,2`
  - Gleitkommazahlen: `3.1415926535897931`
  - Text: `"Python"` und `u"Text"` (in Python 3 sind alle Strings Unicode)
  - `False` und `None` (einfach mal ausprobieren)

Grundlegende Datentypen

- `int` für Ganzzahlen: `0,1,2`
- `float` für Gleitkommazahlen: `3.1415926535897931`
- `str` und `unicode` für `"Text"` und `u"Text"`
- “Besondere” Datentypen sind:
- `bool` für `True` oder `False`
- `None` (noch) kein Wert zugewiesen

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.
- Starte einen geeigneten Editor (notepad++ oder den ActivePython-Editor)
- Speichern mit Endung "`.py`" (Beispiel "`script.py`")
- Ausführen von der Kommandozeile mit "`python script.py`"

Textausgabe

```
1 from __future__ import print_function
2 print("Hello world!")
3 print("Hello " + " world!")
4 print("Hello",world)
5 print(1,"plus",2,"is",1+2)
```

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

Zeile 2 Wie Zeile 1, allerdings mit Stringkonkatenation

Zeile 3 Interpolierte Textausgabe: gibt eine Zeichenkette mit einem Platzhalter `%s` aus. Der Platzhalter wird in der Ausgabe durch den Wert `"world"` ersetzt.

Zeile 4 Interpolierte Ausgabe mit mehreren Platzhaltern: die Platzhalter `%s` werden entsprechend ihrer Reihenfolge ersetzt

Texteingabe

```
1 from __future__ import print_function
2 input = raw_input()
3 print(input)
```

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

```
1 from __future__ import print_function
2 name = raw_input("What is your name? ")
3 print("Hello",name,"!")
```

- `raw_input()` wird hier mit einem zusätzlichen Argument `"What is your name? "` aufgerufen. Die Funktion gibt so zunächst das Argument auf dem Bildschirm aus und wartet dann auf eine Benutzereingabe. So lassen sich einfache Benutzereingaben umsetzen

Materialien

- **Tutorials:** `http://www.learnpython.org/` (interaktiv), `http://docs.python.org/tutorial/introduction.html`
- **Dokumentation** zum Python-Core 2.7.4 auf `http://docs.python.org/2/` oder `pydoc $TERM` auf der Shell.
- **HILFE!!** Gute Q&A-Site für Programmierfragen `https://stackoverflow.com`
- Visualizer: `http://pythontutor.com/visualize.html`

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.