

Einführung in die Programmierung

- Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm
- Hochschule Harz
- FB Automatisierung und Informatik
- mwilhelm@hs-harz.de
- <http://www.miwilhelm.de>
- Raum 2.202
- Tel. 03943 / 659 338

·Inhalt

- **Einführung**
- Einfache Sprachkonzepte
 - Datentypen
 - Variablen
 - Zuweisungen
 - Operatoren
- Verzweigungen und Schleifen
- Methoden
- struct / Objekte
- User Interface mit TKinter

Überblick über Python

- Python ist eine Interpreter, High-level Programmiersprache. Es kann für alles eingesetzt werden.
- In den 1980er von Guido van Rossum entwickelt und im Jahr 1991 veröffentlicht.
- Python's Design Philosophie zielt darauf ab, übersichtlichen Code mit wenigen Whitespaces zu schreiben.
- Es kann benutzt werden für kleine und große Projekte. Inklusive Quantenrechner.
- Python hat eine dynamische Typverwaltung und einen Garbage-Collector.

Überblick über Python

- Es unterstützt Multiple Programming Paradigms mit Strukturen, prozeduralen Anweisungen, Object-orientierte Anweisungen und funktionale Sprachelemente. (Mathematische Definition der Anweisungen)
- Python wurde Anfang der 1980 Jahre als Nachfolger der ABC-Sprache entwickelt.
- Python 3.0 wurde 2008 veröffentlicht.
- Python Interpreter sind für viele Plattformen verfügbar.
- Python ist so konzipiert, dass es ohne Probleme weitere Module integrieren kann:
 - Numpy, Pandas

Überblick über Python

- Python ist eine klar strukturierte AllzweckProgrammiersprache
- Unterstützt verschiedene Programmiermodelle
 - Imperativ
 - Funktional (Mathematische Notation)
 - Objektorientiert
- Umfangreiche Standard-Bibliothek
 - Leistungsfähige Module für viele Anwendungen
- Betriebssystemunabhängig und portabel
 - Interpreter existieren für viele unterschiedliche Plattformen
- Python-Code ist sehr gut lesbar und wartbar
 - Einfache und elegante Syntax durch Einrückungen

Guido van Rossum



Quelle: Wikipedia

Zen_of_Python

- **Beautiful is better than ugly.**
- **Explicit is better than implicit.**
- **Simple is better than complex.**
- Complex is better than complicated.
- **Flat is better than nested.**
- Sparse is better than dense. (Spärlich/Wenig vs. dicht)
- **Readability counts.**
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- **Errors should never pass silently.**
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one—and preferably only one—obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than right now.[a]
- **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.**
- Namespaces are one honking great idea—let's do more of those!

Syntax und Semantik

- Python ist einfach zu lesen und zu verstehen.
- Die Formatierung ist sichtbar ordentlicher als bei anderen Sprachen,
- Python benutzt gegenüber anderen Programmiersprachen mehr englische Wörter, während andere Klammern und Striche benutzen.
- Die Integer-Zahlen können beliebig lang werden!
- Die Rückgabe von Daten kann auch mit Tupeln geschehen (sehr hilfreich).

Entwicklung / Terminologie

■ Ingenieurwissenschaften

- Ingenieure entwickeln mit wissenschaftlichen Methoden und Werkzeugen:
 - Produkte (Autos, Smartphones, Fernseher etc.)
 - Verfahren zur Automatisierung von Prozessen (Roboter etc).
 - Selbstfahrende Autos (KI)

■ Informatiker

- Informatiker entwickeln mit wissenschaftlichen Methoden und Werkzeugen:
 - Computerprogramme
 - Dienen den Ingenieuren als Basis.
 - Excel / Winword / Taschenrechner / Facebook etc.

■ Computerprogramme brauchen:

- Problem
- Beispiele
- Ein Algorithmus (benannt nach Al Chwarismi, geb. ca. 780 n. Chr.)
 - Eine Arbeitsanleitung zur Lösung des Problems
- Ein exaktes und fehlerfreies Computerprogramm
- Ein Übersetzer (Maschinencode)

Entwicklung / Terminologie

■ Arbeitsanleitungen

- Wechseln eines Autoreifens
- Bestellen einer Pizza
- Aktives Spielen von Musik (Noten, Reihenfolge)
- Konzert (Dirigent, Musiker, Noten)
- Erstellen einer Wertetabelle einer Funktion: x/y -Tabelle

■ Aufbau

- Menge von unterschiedlichen Anweisungen an unterschiedlichen Personen

■ Charakteristika

- Anweisungssequenzen (eine bis mehrere hintereinander)
- Bedingte Anweisungen (Reifenwechsel: Ist der Reifen heil?)
- Anweisungsschleifen (Spiel die Noten bis zum Ende des ersten Satzes).
- Zutaten (neuer Reifen)
- Exakte vs. schwammige Formulierungen
 - Zieh die Schrauben fest an
 - Spiel die Noten schnell, langsam.

Beispiele für Algorithmen

■ Addiere zwei Zahlen

- Schreibe die erste Zahl auf einem Zettel
- Schreibe die zweite Zahl auf einem Zettel
- Addiere beide Zahlen

Beispiele für Algorithmen

- **Reifenwechsel**

- ???

Beispiele für Algorithmen

■ Reifenwechsel

- Das Auto parken
- Den Motor ausstellen
- Die Handbremse feststellen.
- Gang reinlegen (Kupplung?).
- Holen des Wagenhebers (wo ist er?).
- Lösen der Schrauben (welche, an allen Reifens?).
- Beim kaputten Rad den Wagenheber befestigen (am Rad?).
- Das Auto hochbocken.
- Lösen der Schrauben (es fehlt der Kreuzschlüssel!).
- Entfernen des kaputten Reifens (wie, wo).
- Aufstecken des neuen Reifens.
- Festziehen der Schrauben (leicht fest , sehr präzise!).
- Den Wagenheber runterlassen.
- Festziehen der Schrauben (nun fest, sehr präzise).
- Verstauen des Wagenhebers.
- Verstauen des kaputten Reifens.

Beispiele für Algorithmen

■ Suche das Buch mit den meisten Seiten

- Gehe zum Regal
- Nimm das erste Buch
- Schreibe die Seitenzahl an eine Tafel
- Nimm das zweite Buch
- Ist die Seitenzahl größer als die an der Tafel?
 - Ersetze die Zahl an der Tafel
- Nimm das dritte Buch
- Ist die Seitenzahl größer als die an der Tafel?
 - Ersetze die Zahl an der Tafel
- Nimm das vierte Buch
- Ist die Seitenzahl größer als die an der Tafel?
 - Ersetze die Zahl an der Tafel
- Gib es noch ein Buch?
 - Wenn nicht, dann Ausgabe der Seitenzahl
 - Sonst zum nächsten Buch

Beispiele für Algorithmen

- **Gegeben 100 Zettel mit jeweils einer Nummer**
- **Suche die kleinste Zahl**
 - Gehe zum Tisch mit den Zettel
 - Nimm den ersten Zettel in die Hand
 - Schreibe die Zahl an eine Tafel
 - Legt den Zettel zur Seite

 - Nimm den zweiten Zettel in die Hand
 - Ist die Zahl kleiner als die an der Tafel?
 - Ersetze die Zahl an der Tafel

 - Nimm den dritte Zettel in die Hand
 - Ist die Seitenzahl kleiner als die an der Tafel?
 - Ersetze die Zahl an der Tafel

 - Gib es noch einen Zettel?
 - Wenn nicht, dass Ausgabe der kleinsten Zahl
 - Sonst zum nächsten Zettel

Elemente eines Algorithmus

- **Direkte Anweisung**
- **Initialisierung (Kollege, gib mir die 100 Zettel mit den Zahlen)**
- **Holen eines Wertes (lese Zahl auf dem Zettel)**
- **Zuweisung (Schreibe Zahl an die Tafel)**
- **Abfragen (Ist Zahl kleiner/größer/gleich)**
- **Schleifen (wenn es noch Zettel gibt $\Rightarrow$ nochmal lesen)**
- **Ende mit Ausgaben**

- **Zusätzlich:**
 - Delegiere (Anruf an einen Kollegen: **Ist 5 kleiner als 7?**)
 - An den Beifahrer: Hole das Ersatzrad aus dem Kofferraum.

Definition des Algorithmus

■ Definition

- Arbeitsanleitung zum Lösen eines Problems, die so präzise formuliert sein muss, dass sie von einem Computer ausgeführt werden kann.

■ Ausführung von Algorithmen

- Menge von unterschiedlichen Anweisungen an unterschiedlichen Personen

■ Formulierung von Algorithmen

- Anweisungssequenzen (eine bis mehrere hintereinander)
- Bedingte Anweisungen (Reifenwechsel: Ist der Reifen heil?)
- Anweisungsschleifen (Spiel die Noten bis zum Ende des ersten Satzes).
- Zutaten (neuer Reifen)
- Exakte vs. schwammige Formulierungen
 - Zieh die Schrauben fest an
 - Spiel die Noten schnell, langsam.

■ Computer

- Ein Computer ist extrem **schnell**
- Ein Computer ist extrem **doof** (Die Anweisungen muss exakt sein!)