Algorithmik und Programmieren

Brückenkurs

Michael Wolf

22.9.2014 - 26.9.2014

Table of Contents

Termine

Der Kurs findet an folgenden Terminen statt:

- Montag 22.9.14 11:15 - 13:00
- Dienstag 23.9.14 14:15 - 17:00
- Mittwoch 24.9.14 11:15 - 16:00
- Donnerstag 25.9.14 11:15 - 13:00
- Freitag 26.9.14 14:15 - 16:00

Für wen ist dieser Kurs

Und was lernt man hier

Der Kurs ist für:

- Programmieranfänger
- Informatikstudierende und die es noch werden wollen

Was lernt man hier?

- Grundlagen der Programmiersprache Python
- Grundlagen der Programmierung
- Implementierung von einfachen Algorithmen

Ablauf

Jede Einheit umfasst zwei Teile

Theorie Besprechung der Konzepte und Hintergründe, die benötigt werden um Programme zu schreiben.

Praxis Vertiefung der Theorie anhand zahlreicher Beispiele, die Sie alleine oder in Zusammenarbeit mit ihrem Sitznachbarn lösen sollten.

Ablauf der Übungen

Die praktischen Übungen machen den Haupteil dieser Lehrveranstaltung aus. Am effektivsten lernt man Programmieren, indem man es macht!
Zu jeder Übungseinheit gibt es einen Übunszettel, den man alleine oder mit dem Sitznachbarn löst. Die Aufgaben beziehen sich auf die Konzepte die zuerst im Theorieteil der Übung besprochen werden.

Was ist Python?

- wurde Anfang der 90ger von Guido van Rossum entwickelt
- legt großen Wert auf Programmlesbarkeit (kurzer, einfacher Syntax)
- Plattformunabhängig
- interpretierte Hochsprache
- sehr beliebt in der Hacker Community
- Frei und Quelloffen

Resourcen

- python homepage `https://www.python.org/`
- youtube
- `http://www.python-kurs.eu/`
- A Byte of Python, kostenlos zum download hier: `http://swaroopch.com/notes/python/`
- die Hilfe im Interpreter *help()*

Und los...

"Manchmal muss man rennen, bevor man laufen kann"

Tony Stark zu Jarvis, Iron Man

- Konsole öffnen
- python eingeben
- 1+1 eingeben
- schauen was passiert
- Abbrechen mit Ctrl+D oder exit() eingeben

Was ist da gerade passiert?

Das war der Interpreter, 1 und 1 wurden erfolgreich addiert und das Ergebnis war 2
Es gibt zwei Möglichkeiten ein Python Programm auszuführen:

- Interpreter - Nach jeder Zeile Code sieht man sofort das Ergebnis.
- Ein Programm kann man aber auch mittels *python programm.py* ausführen. Alle Kommandos in *programm.py* werden so ausgeführt, als ob man Zeile für Zeile im Interpreter eingibt.

Variablen

Was ist eine Variable

- Ein Programm verarbeitet **Daten**, die in sogenannten **Variablen** abgelegt werden.
- Ein Programm legt die Ergebnisse wieder in solchen **Variablen** ab.
- Eine Variable hat einen **Variablennamen** und einen **Typ**

Variablennamen

Gültige Variablennamen sind eine beliebige Kombination aus folgenden Zeichen: a-z, A-Z, 0-9 und '_'.
Ausnahmen:

- Variablennamen dürfen nicht mit Ziffern beginnen
- Reservierte Schlüsselwörter and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

Variablennamen sollten selbsterklärend sein und englische Namen haben
Zum Beispiel: der Vorname sollte in eine Variable *firstname* gespeichert werden und nicht in eine Variable *f*

Datentypen

Python bietet Datentypen für die gängigsten Anwendungen:

- Strings (Zeichenketten): 'This␣is␣text!', "That's␣more␣text."
- Integer (Ganzzahlen): 1, 4294967296, 010, 0xFF, 0b100
- Float (Gleitkommazahlen): 1.35, 1.56e-5 ($1.56 * 10^{-5}$)
- Complex (Komplexe Zahlen): $2 + 3j$
- Bool (Wahr Falsch): True

Variablen cont.

Variablen haben in python keinen statischen Typ, d.h. es muss nicht vorher angegeben werden, welchen Typ sie haben und der Typ kann sich zur Laufzeit ändern.
Variablen haben aber einen dynamischen Typ, der bestimmt welche Operationen für diese Variable zulässig sind!

```
1 >>> a = 2
2 >>> a = "test"
3 >>> a + 3
4 Traceback (most recent call last):
5   File "<stdin>", line 1, in <module>
6 TypeError: cannot concatenate 'str' and
    'int' objects
```

Operatoren

Über Operatoren erhält man die Möglichkeit, z.B. Variablen mit Inhalten zu belegen oder die in ihnen gespeicherten Werte zu verändern.

Operatoren cont.

Operator	Erklärung	Beispiel
+, -	Addition, Subtraktion	10 - 3
*, /, %	Mult., Division, Rest	27 % 7 $\rightarrow$ 6
+x, -x	Vorzeichen	-3
~x	Bitweises Not	3 - 4 Ergebnis: -8
$<, <=, >, >=, !=, ==$	Vergleichsoperatoren	2 <= 3 $\rightarrow$ WAHR
or, and, not	ODER, UND, NICHT	(a or b) and c
**	Exponentiation	2**3 $\rightarrow$ 8
in	"Element von"	1 in [3, 2, 1]
|, &, ^	Bitoperatoren or, and, xor	6 $\hat{}$ 3
<<, >>	Shiftoperatoren	6 << 3

Operatioren Beispiele

```
1  a = 2
2  b = 3
3  c = a*b
4
5  print a, "*", b, "=", c
6
7  c = a**b
8  print a, "hoch", b, "ist", c
9
10 if a > b:
11     print "a ist groesser"
12 else:
13     print "a ist kleiner"
```

Kontrollstrukturen

- Bisher wurden die einzelnen Zeilen eines Programms sequentiell abgearbeitet.
- Sehr oft will man aber bestimmte Anweisungen auswählen oder wiederholen.
- Kontrollstrukturen definieren die Reihenfolge, in der Berechnungen durchgeführt werden.

Anweisungen und Blöcke

Anweisung

- Eine Anweisung ist zum Beispiel: x * 3
- anders als in manchen Programmiersprachen muss eine Anweisung nicht von einem Semicolon gefolgt werden

Block

- in Python bestimmt die Einrückung den Code Block
- Ein Block ist syntaktisch äquivalent zu einer einzelnen Anweisung.
- Blöcke kann man auch ineinander schachteln, indem man nochmals mittels Tabulator einrückt
- im Interpreter 4 Whitespaces benutzen, anstatt Tabulator
- Beispiele folgen noch.

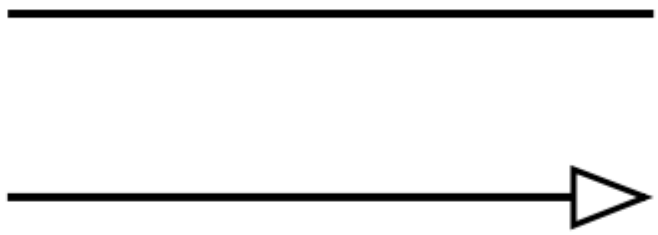
Flussdiagramme

nach wikipedia

- Ein Flussdiagramm dient zur grafischen Darstellung eines Algorithmus.
- wird auch als Programmstrukturplan bezeichnet
- kann auch zur Darstellung von Abläufen verwendet werden

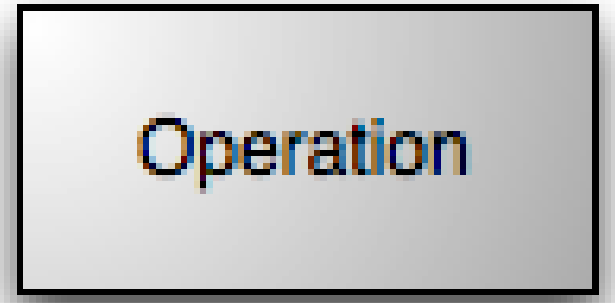
Elemente eines Flussdiagrams

Start und End Zustand



Verbindungslinien, Pfeilrichtung gibt Programmfluss an

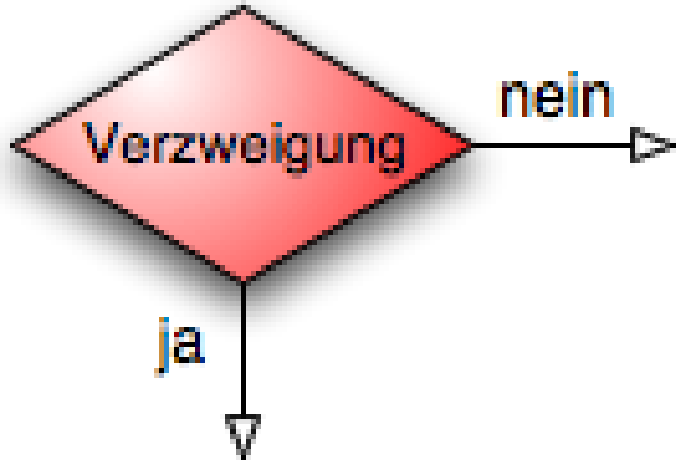
Elemente eines Flussdiagrams



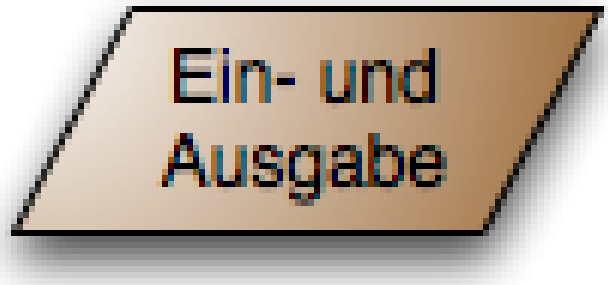
Eine Anweisung

Start eines Unterprogramms, wartet bis zur vollständingen

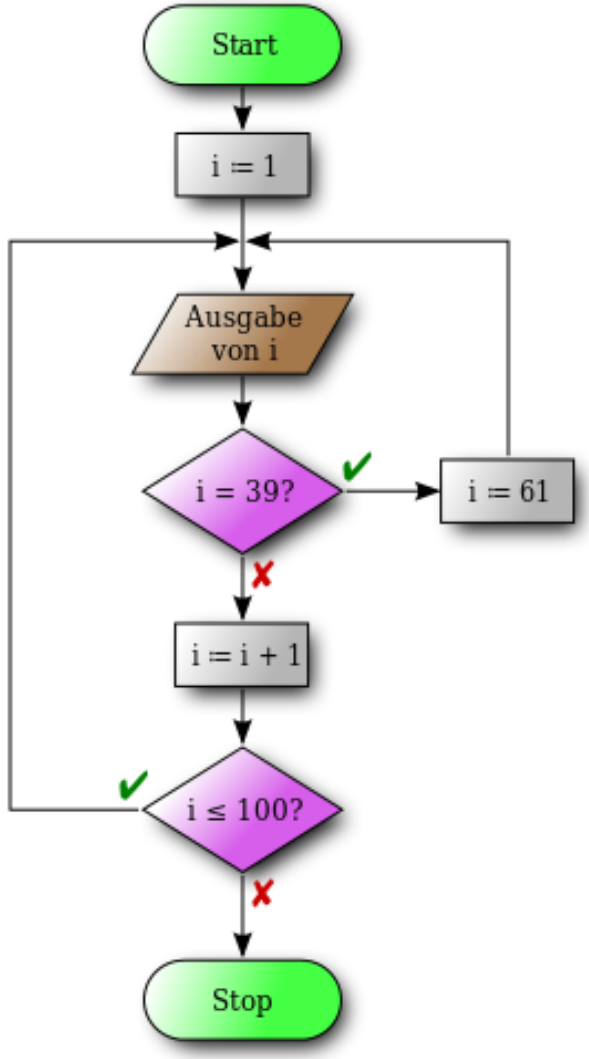
Elemente eines Flussdiagrams



Eine Verzweigung bzw ein Ort an dem eine Entscheidung über den Programmfluss gefällt werden muss



Flussdiagramm Beispiel



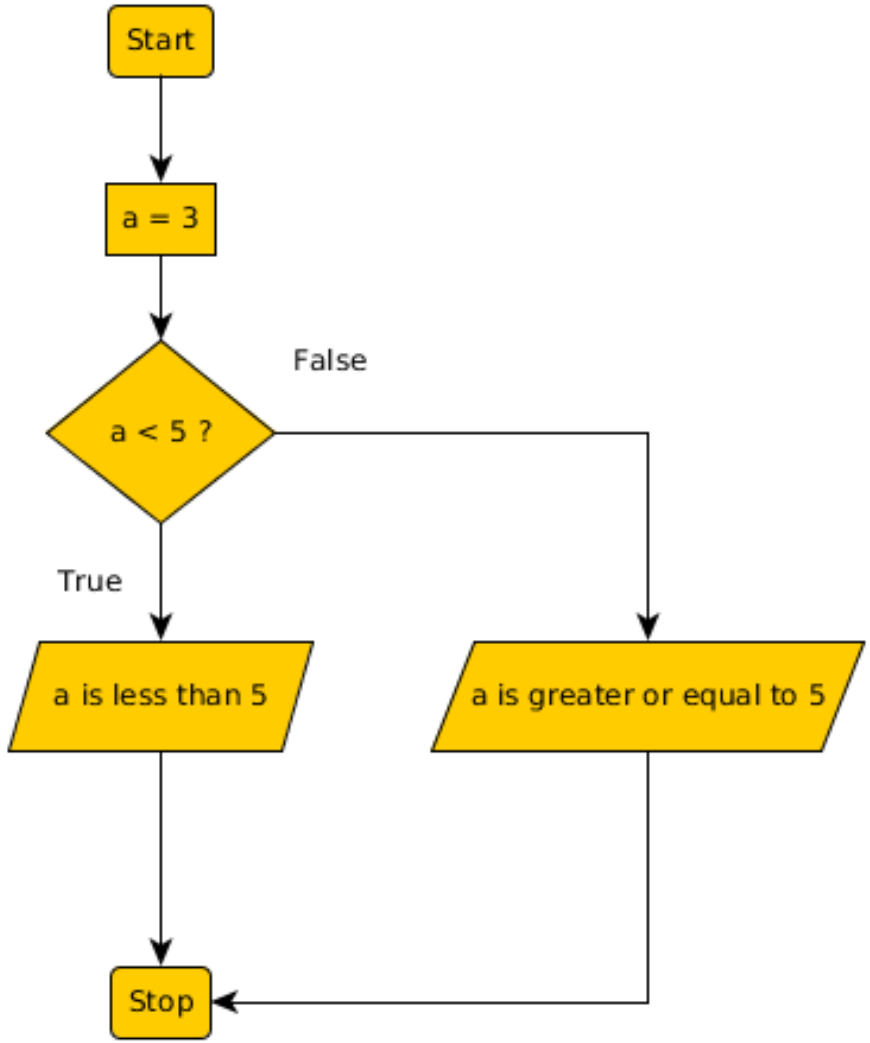
Verzweigung mit if

Allgemeine Form:

```
1 if condition_1:
2 >>>print "condition 1 evaluated to true"
3 elif condition_2 and condition_3:
4 >>>print "condition 2 and condition 3
     evaluated to true"
5 elif condition_N:
6 >>>print "condition N evaluated to true"
7 else:
8 >>>print "none of them is true"
```

- >>> steht für einen Tabulator
- Der `else` und die `elif` Zweig(e) ist optional
- die Anweisungen nach den if Statements müssen in einem eigenen Code Block liegen

Flussdiagrammdarstellung



Schleifen mit while

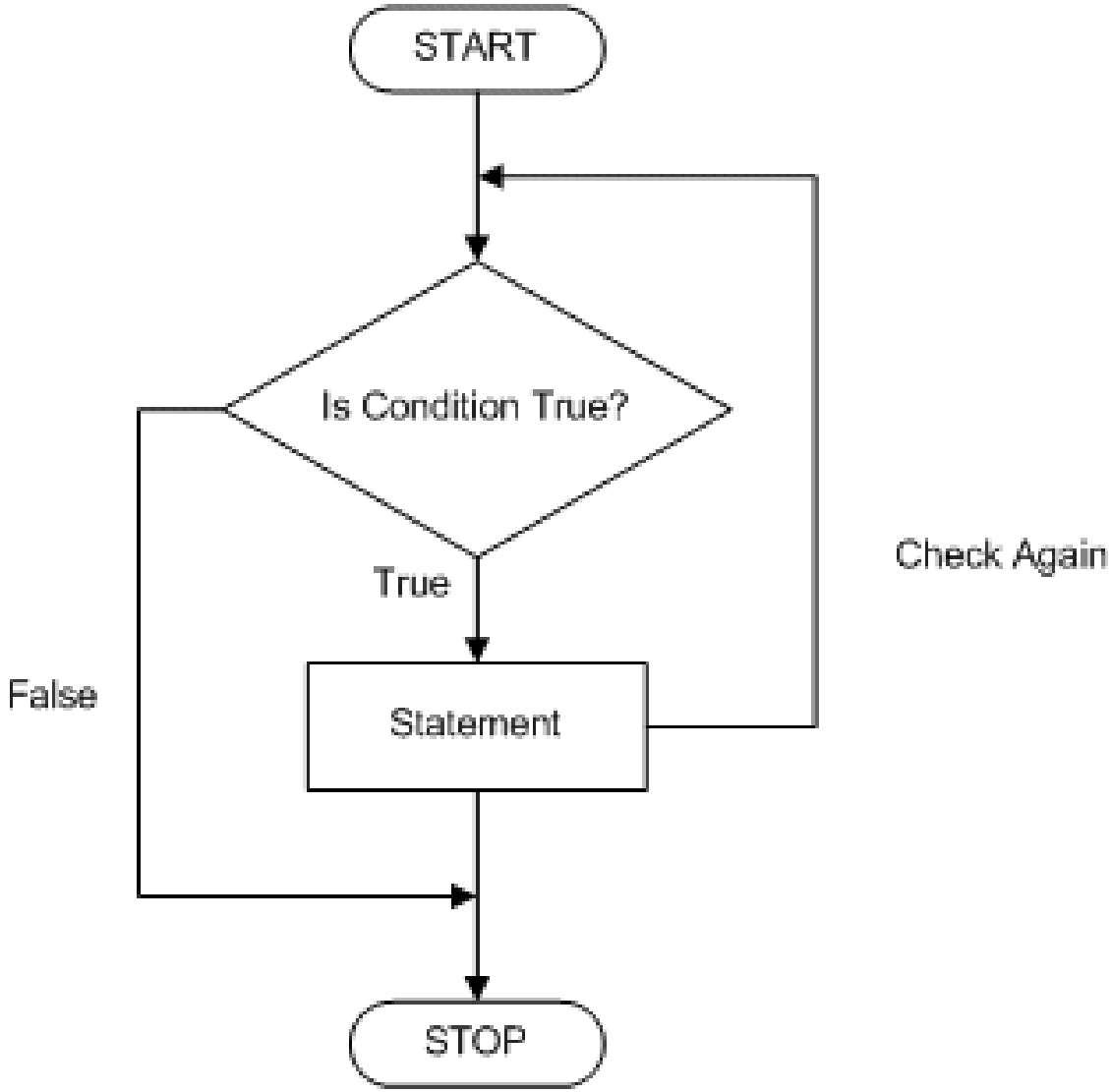
```
1 while condition:
2   print "i am going to repeat myself as
        long as condition is true"
3 else:
4   print "condition is false"
```

- Zunächst wird der Ausdruck (**condition**) ausgewertet.
- Ist der Ausdruck WAHR, so wird die nachfolgende Anweisung ausgeführt.
- Anschließend wird wieder der Ausdruck ausgewertet.
- ist der Ausdruck nicht mehr WAHR können optional im `else` Block Anweisungen folgen

Wichtig

- Beliebiger Teil des Ausdrucks muss im Schleifenrumpf (Anweisung) verändert werden (ansonsten erhält man eine Endlos-Schleife).

Flussdiagramm while



Schleifen mit while - Beispiel

```
1  a = 0
2  while a < 3:
3      print a
4      a = a + 1
5  else:
6      print "done"
```

Schleifen mit while - Beispiel

```
1 a = 0
2 while a < 3:
3     print a
4     a = a + 1
5 else:
6     print "done"
```

produziert die Ausgabe:

```
1 0
2 1
3 2
4 done
```

Schleifen mit for

for-Schleifen werden traditionell für Ausdrücke verwendet, die sich N mal wiederholen. Das Beispiel benutzt die Zähler Variable *a* welche von 0 bis 2 läuft, 3 ist hier die Obere grenze, welche nicht ereicht wird

```
1 for a in range(0,3):
2     print a
```

Schleifen mit for

for-Schleifen werden traditionell für Ausdrücke verwendet, die sich N mal wiederholen. Das Beispiel benutzt die Zähler Variable *a* welche von 0 bis 2 läuft, 3 ist hier die Obere grenze, welche nicht ereicht wird

```
1 for a in range(0,3):
2     print a
```

produziert die Ausgabe:

```
1 0
2 1
3 2
```

Der for-Schleife kann wie bei der while Schleife ein else block folgen

break und continue

Eine Schleife muss nicht immer bis zum Ende durchgeführt werden. Mit **break** kann wird die Schleife sofort beendet.

```
1  a = 0
2  while a < 10000:
3      if a == 5:
4          break;
5      print(a)
6      a = a + 1
```

break und continue

Eine Schleife muss nicht immer bis zum Ende durchgeführt werden. Mit **break** kann wird die Schleife sofort beendet.

```
1 a = 0
2 while a < 10000:
3     if a == 5:
4         break;
5     print(a)
6     a = a + 1
```

produziert die Ausgabe:

```
1 0
2 1
3 2
4 3
5 4
```

break und continue

Hingegen kann mit **contintue** wieder an den Schleifenanfang gesprungen werden.

```
1  a = 0
2  while a < 10:
3      if a % 3 == 0:
4          a = a + 1
5          continue;
6      print(a)
7      a = a + 1
```

break und continue

Hingegen kann mit **contintue** wieder an den Schleifenanfang gesprungen werden.

```
1 a = 0
2 while a < 10:
3     if a % 3 == 0:
4         a = a + 1
5         continue;
6     print(a)
7     a = a + 1
```

produziert die Ausgabe:

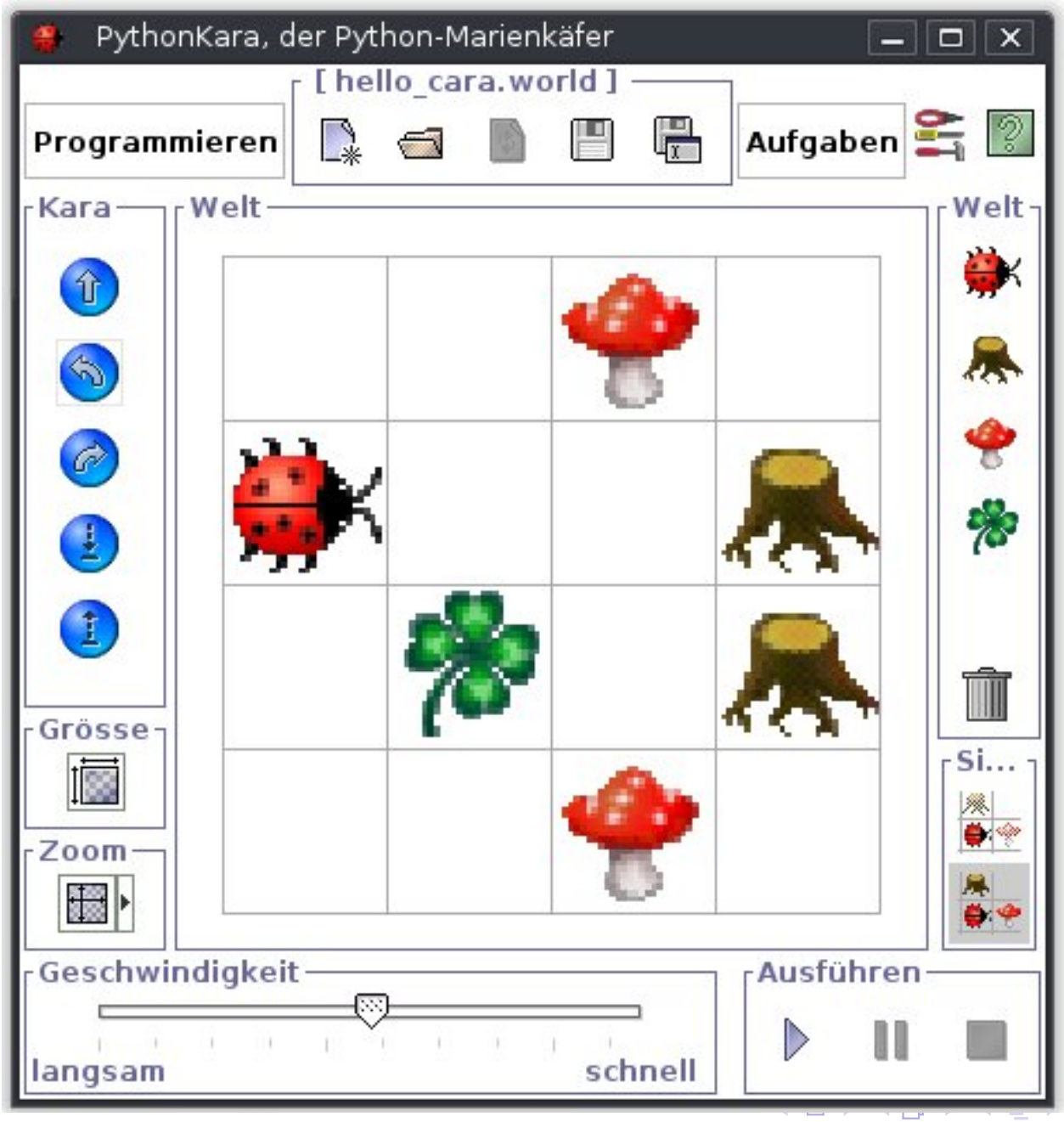
```
1 1
2 2
3 4
4 5
5 7
6 8
```

PythonKara – Programmieren mit Python

Einführung

- bietet eine einfache Einführung in die Programmiersprache Python
- man kann den eigenen Code anhand der Aktionen von Kara nachverfolgen
- Entwickelt von Raimond Reicherts
- ist Preisträger des European Academic Software Award 2002
- macht Spaß!!
- zu finden auf `http://www.swisseduc.ch/informatik/karatojava/index.html`

Die Oberfläche



Elemente in Karas Welt

- **Kara** der Programmierbare Käfer
- **Bäume** Kara kann nicht durch/über Bäume klettern, kann aber erkennen ob ein Baum vor, links oder rechts von ihm steht
- **Fliegenpilz** Kara kann den Fliegenpilz verschieben und erkennen ob er vor ihm steht
- **Kleeblatt** Kara kann erkennen ob er auf einem Kleeblatt steht, er kann ein Kleeblatt legen oder fressen

Wie sage ich Kara, was er tun soll?

Kara kann mit folgenden aufrufen gesteuert werden:

- *kara.move()* - Kara macht einen Schritt in die Richtung, in die er schaut
- *kara.turnRight()* - Kara dreht sich nach rechts
- *kara.turnLeft()* - selbiges wie oben nur anders herum
- *kara.putLeaf()* - Kara legt ein Kleeblatt auf das Feld auf dem er steht
- *kara.removeLeaf()* - Kara frisst ein Kleeblatt, wenn auf dem Feld, auf dem er sich befindet, eines liegt

ACHTUNG Wenn Kara über den Weltrand rausläuft kommt er auf der gegenüberliegenden Seite wieder raus!

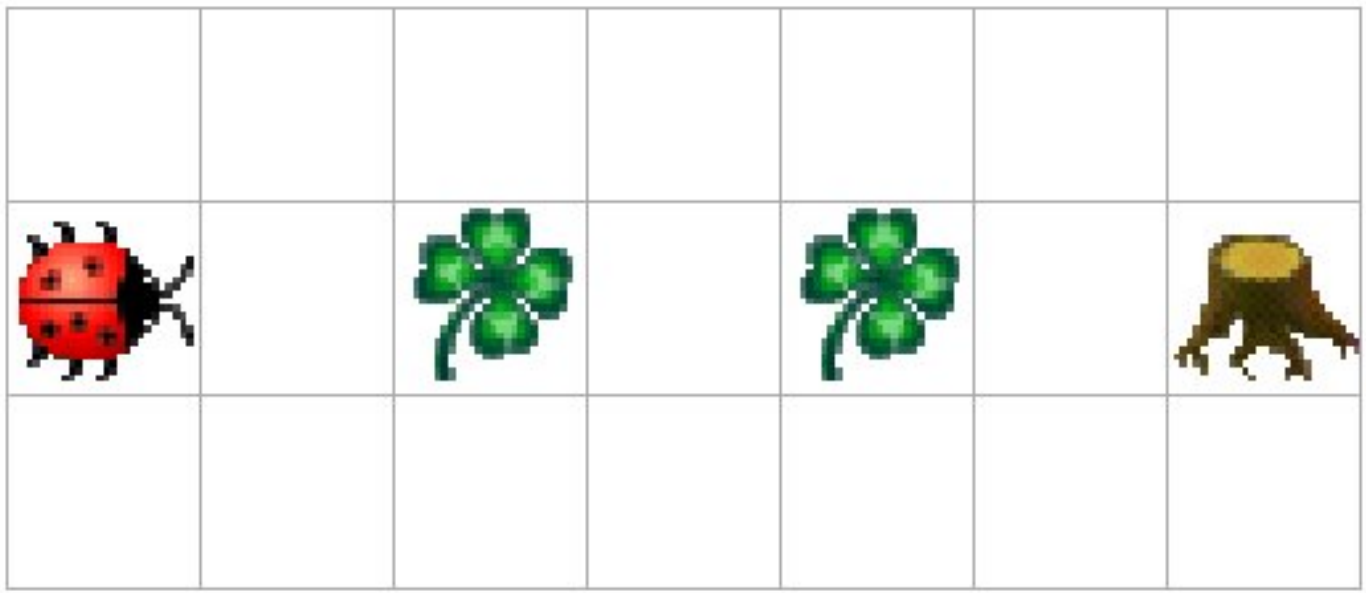
Wie sieht Kara die Welt?

Kara kann seine Umgebung mit folgenden Sensoren überwachen:

- *kara.treeFront()* - ist ein Baum vor mir?
- *kara.treeLeft()* - ist ein Baum links neben mir?
- *kara.treeRight()* - ist ein Baum rechts neben mir?
- *kara.onLeaf()* - stehe ich auf einem Blatt?
- *kara.mushroomFront()* - ist vor mir ein Fliegenpilz?

Beispiel

Kara soll bis zum nächsten Baumstumpf gehen und dabei alle Kleeblätter fressen.



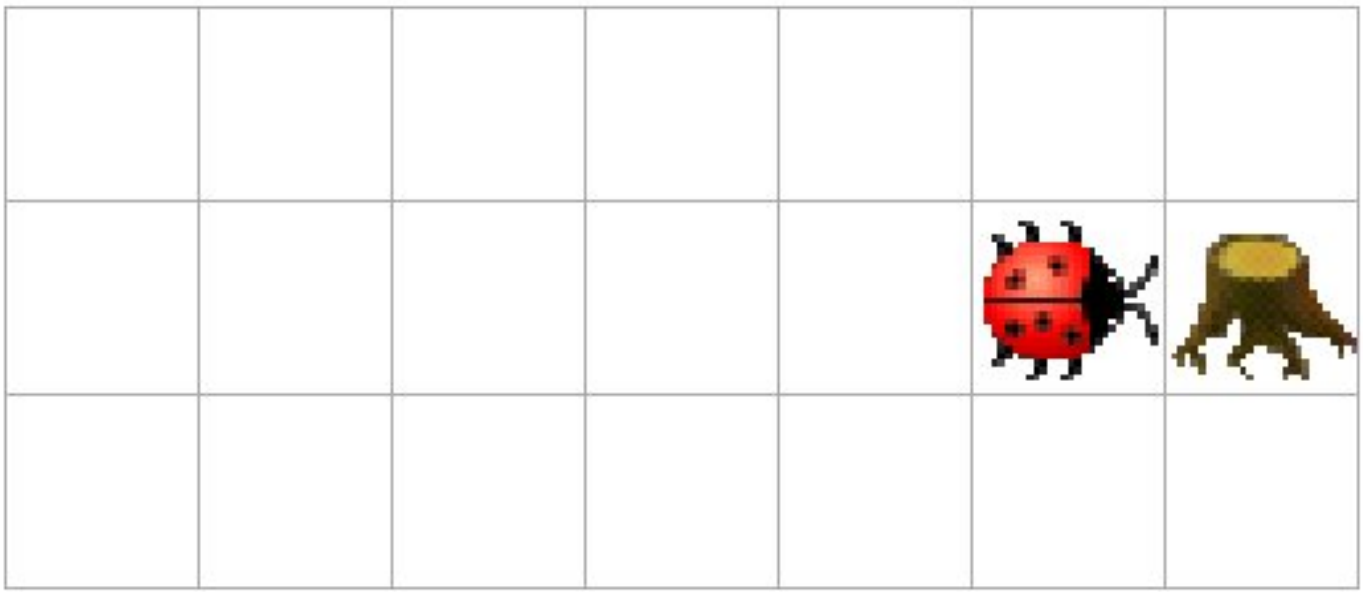
Beispielcode

Mit diesem Code geht **Kara** bis zum nächsten Baumstumpf und frisst dabei alle Kleeblätter

```
1 #solange kein Baum vor Kara ist
2 while not kara.treeFront():
3 #ist kara auf einem Blatt?
4   if kara.onLeaf():
5     kara.removeLeaf() #hmm lecker
6   kara.move() #gehe einen schritt
```

Beispiel

Kara: “LECKER!!”



Woher bekomme ich Kara

- **Kara** kann hier `http://www.swisseduc.ch/informatik/karatojava/pythonkara/classes/pythonkara-x.jar` runtergeladen werden.
- Gestartet wird Kara in einer Konsole mit *java -jar pythonkara-x.jar*

Funktionen

- Funktionen haben die Aufgabe, Teile eines Programms unter einem eigenen Namen zusammenzufassen.
  - Damit kann man dann diesen Programmteil mit einem Namen aufrufen.
  - Beim Aufruf kann man Parameter mitgeben.
  - Funktionen können auch Ergebnisse zurückliefern.
- Funktionen sind ein Mittel zur Strukturierung eines Programms.
  - Bei einem großen Programm kann man nicht alle Anweisungen in einem Block zusammenfassen, da sonst der Überblick verloren geht.
  - Ein Programm sollte mit Hilfe von Funktionen in modulare Teile aufgeteilt werden.

Funktionen cont.

- Funktionen sind ein Mittel zur Wiederverwendung.
  - Eine Funktion kann an mehreren Stellen (mit unterschiedlichen Parametern) aufgerufen werden.
  - Die Funktion selbst ist nur einmal im Programm vorhanden und verkürzt daher den Programmcode

Funktionsdefinition

```
1 def function(parameter1, ... parameter_n):
2 >>>#do something
3 >>>return calculation
4
5 #call function
6 result = function(a, b, ... ,z)
```

Funktionen - Beispiel

```
1  def mul(a, b):
2      return a*b
3
4  def printABC():
5      print("abcdefghijklmnopqrstuvwxyz")
6
7  c = mul(2,3)
8  print(c)
9  printABC()
```

Funktionen - Beispiel

```
1 def mul(a, b):
2     return a*b
3
4 def printABC():
5     print("abcdefghijklmnopqrstuvwxyz")
6
7 c = mul(2,3)
8 print(c)
9 printABC()
```

erzeugt Ausgabe:

```
1 6
2 abcdefghijklmnopqrstuvwxyz
```

Listen

Bisher haben wir uns mit einzelnen Variablen und Konstanten beschäftigt. Datenstrukturen erlauben es große Datenmengen zusammenzufassen, und zu verarbeiten.
Die einfachste Datenstruktur in Python ist die Liste. Eine Liste ist eine geordnete (geordnet heißt nicht zwangsläufig sortiert!), endliche Sammlung von Elementen beliebiger Datentypen.

Listen Beispiel

```
1 >>> list = [3,42,1003]
2 >>> list
3 [3, 42, 1003]
4
5
6 list = [3,42,1003]
7
8 for i in list:
9     print i
```

Ausgabe:

Listen Beispiel

```
>>> list = [3,42,1003]
>>> list
[3, 42, 1003]


list = [3,42,1003]

for i in list:
    print i
```

Ausgabe:

```
3
42
1003
```

Listen Beispiel II

Zeichenketten verhalten wie Listen:

```
1 >>> for c in city:
2 ...     print c
3 ...
```

Ausgabe:

Listen Beispiel II

Zeichenketten verhalten wie Listen:

```
>>> for c in city:
...         print c
...
```

Ausgabe:

```
I
n
n
s
b
r
u
c
k
```

Operationen auf Listen

- `len([1, 2, 3])` Länge der Liste = Anzahl der Elemente
- `[1, 2, 3] + [4, 5, 6]` Konkatenation
- `['Hi!']  * 4` Wiederholung
- `3 in [1, 2, 3]` Membership
- `for x in [1, 2, 3]:  print x`, Iteration
- `list[n]` greife auf das n-te Element in Liste zu
- `list[n:m]` greife auf die alle Elemente von n bis m zu
- `list.append(var)` fügt die Variable `var` an die Liste

Algorithmen

nach Wikipedia

Was ist ein Algorithmus?

> *Ein Algorithmus ist eine eindeutige Handlungsvorschrift zur Lösung eines Problems oder einer Klasse von Problemen. Algorithmen bestehen aus endlich vielen, wohldefinierten Einzelschritten.*

Beispiel - Bubblesort

- `j = len(list)-1`
- zwei Elemente `list[i]` und `list[i+1]` einer Liste werden miteinander verglichen
- ist `list[i]` größer als `list[i+1]` vertausche die Elemente in der Liste
- `i = i+1`
- wiederhole diese Schritte bis `i == j`
- `j = j-1`
- starte erneut bei `i=0` solange bis `j == 0`

Demo

Eingabe per Kommandozeile

```
1  x = raw_input("enter a number: ")
2  print x, " is type of ", type(x)
3
4  x = int(x)
5
6  print x, " is type of ", type(x)
```