Python - Grundlagen

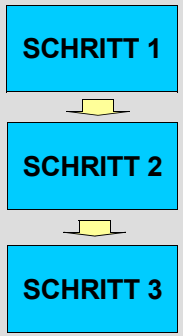
Albert Weichselbraun

Inhalt

- Programmierung - Grundlagen
- Python – Grundlagen
- Datentypen und Operatoren
- Kontrollstrukturen
  Schleifen
  Verzweigungen
- Funktionen

Programmierung

- Programmierung ist die Planung einer Abfolge von Schritten (Instruktionen), nach denen ein Computer handeln soll.



Lebenszyklus von Programmen

1. Problemlösung
2. Implementierung
3. Wartung

Problemlösung

- **Analysieren** des Problemes und **Spezifizierung** einer Lösung
- Entwicklung einer generellen Lösung **(Algorithmus)** um das Problem zu lösen.
- **Überprüfen**, ob die Lösung das Problem wirklich löst.

Problemlösung

Ein Algorithmus ist...

... eine Schritt-für-Schritt Anleitung, um ein Problem in endlicher Zeit zu lösen.

Implementierung

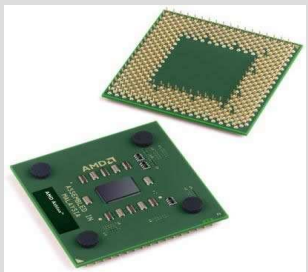
„Die Übersetzung eines Algorithmus in eine **Programmiersprache**“

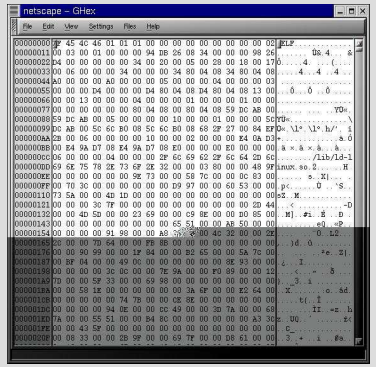
Was ist eine Programmiersprache?

Eine Sprache mit strengen grammatikalischen Regeln, Symbolen und speziellen Wörtern, die verwendet wird, um ein Computer-Programm zu entwerfen.

Implementierung Maschinensprache

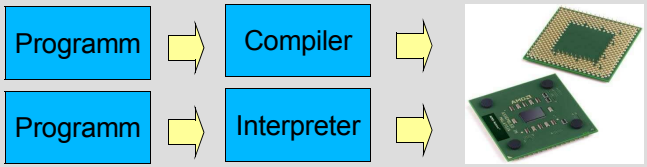
- Das Rechenwerk eines Computers (CPU) versteht nur Maschinensprache:





Implementierung
Höhere Sprachen

- Sind portierbar
- Werden in einer der **natürlichen Sprache** ähnlichen Sprache geschrieben.
- Müssen erst in **Maschinensprache** übersetzt werden.



Implementierung
Compiler contra Interpreter

- **Compiler** übersetzen ein Programm **einmal**, sodass daraus Maschinencode entsteht, den der Prozessor ausführen kann.
- Interpreter übersetzen den Quelltext zur Laufzeit des Programmes.
  - Nachteil: langsamer
  - Vorteil: plattformunabhängig, Zeit für das Komplieren fällt weg

Implementierung
Beispiele für Höhere Sprachen

- BASIC
- C/C++
- Java
- Pascal
- Perl
- **Python**

Python – Typische
Anwendungsgebiete

- Information Retrieval
- CGI- und Web-Anwendungen
- Automatisierung von administrativen Tätigkeiten
- Rapid Prototyping (Forschung, Entwicklung, ...)

Python – Installation testen

- Python Interpreter starten:
  Python2.4.c0 (#2, Jun 14 2006, 22:35:41)
  [GCC 4.1.2 20060613 (Debian 4.1.1-4)]
  Type „help“, „copyright“ or „licence“ ...
  >>>
- Hilfe bekommt man mit
  help(), dir(),
  mit Funktionsname.__doc__
  oder auf der Python Website:
  http://www.python.org

Python - Grundlagen

- Python ist **case-sensitive**
- Die Einrückung der Befehle determiniert den Programmfluss
- Kommentare beginnen mit #
- Alle Datentypen sind **Objekte**
- Verwendung der Sprache im Python Interpreter oder in Skripten

Python als Taschenrechner

- Umgang mit Zahlen:
  2+2, (50-5*6)/4
- Ganze Zahlen (Integer) contra Floats:
  7/3, 7/3.0
- Variablen:
  width=20, height=5*9, width*height
  x = y = z = 0
- Letzte Wert: _
  price = 100, price*1.2, _+200

Arithmetische Operationen

Beispiel	Operator	Ergebnis
4+wert	Addition	Summe von 4 und Wert
zahl1-zahl2	Subtraktion	Differenz von zahl1 und zahl2
nettoPreis*1.2	Multiplikation	Produkt von nettoPreis und 1.2
Kosten/Nutzen	Division	Kosten geteilt durch Nutzen
32%6	Modulo	Ganzzahliger Rest der Division
2**6	Exponent	2 hoch 6

- Beispiel: Kreisfläche
  flaeche = radius**2 * 3.14

Zuweisungsoperatoren

- weisen einer Variable Werte zu

  a=1 a,b = 12, 3
- Berechnung von Variablenwerten:

  a=a+12 a,b = b+3, a-1
- abgekürzte Form:

  a+=12 a*=2

  myString += ' (erledigt) '

Umgang mit Strings

Strings = Abfolge von alphanumerischen Zeichen

- **Eingabe** von Strings:

  'spam eggs', "doesn't", '"Contains " and '... '"
- **Operationen**

  len(word), a.capitalize(), a.upper()
- **Slicing**

  word = 'donaudampfschiffsfahrtsgesellschaft'

  word[0:5] = 'donau', a[-12:]='gesellschaft'

Metazeichen

\n	Zeilenumbruch
\t	Tabulator
\b	Backspace
****	Backslash
\'	' innerhalb eines single-quoted Strings (' ')
\"	" innerhalb eines double-quoted Strings(" ")

Literale Interpretation von „\“: `r'mein Text mit \'`

Umgang mit Strings - Slicing

```
 +---+---+---+---+---+
 | H | a | l | l | o |
 +---+---+---+---+---+
 0   1   2   3   4   5
-5  -4  -3  -2  -1
```

a='Hallo'
a[0:2] -> 'Ha', a[-3:5] = a[-3:] = 'llo'

Übungsaufgabe

- Setzen Sie Variablen mit folgendem Inhalt
  - Ihr Name
  - Ihre Adresse
  - Ihre Lieblingsfarbe
- Geben Sie diese Variable in einem Text mittels print aus.

Erste Schritte Richtung Programmierung

```
# Beispielprogramm

# Fibonacci Reihe

a, b = 0,1

while b < 10:
  print b
  a, b = b, a+b
```

Ausgabe in einer Zeile => Anhängen eines Kommas an den print-Befehl.

Python in Skripten

- Skripte werden wie folgt startfähig gemacht:
  in der **ersten Zeile** das Programm (als Kommentar) angeben

  #!/usr/bin/env python

  Das Skript mit Ausführungsrechten versehen:
  chmod a+x filename.py
  Starten mit ./filename.py

Mein erstes Python Programm

#!/usr/bin/env python

“““ Ein erstes Beispielprogramm “““

print “Hello World!“ # ein weiteres Kommentar

Benutzereingaben

- Benutzereingaben können mit dem Befehl

```
wert_str = raw_input('Prompt: ')
```

eingelesen werden.

- Zahlenwerte müssen vor deren Verwendung konvertiert werden:

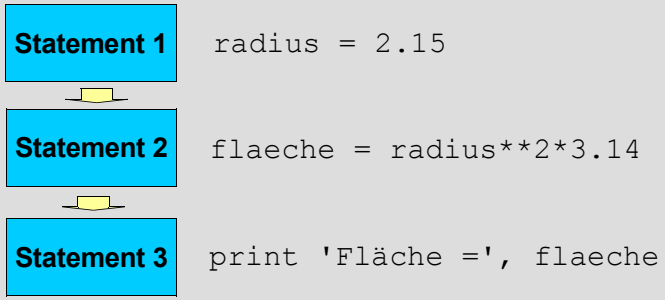
```
radius = float(radius_str)
anzahl = int(anzahl_str)
```

Übungsaufgaben

- Erstellen Sie ein Skript, das nach Eingabe des Bruttopreises einer Ware Nettopreis und MWSt. ausgibt.
- Schreiben Sie ein Programm, das den Benutzer nach Radius und Höhe eines Zylinders fragt und Volumen und Oberfläche des Zylinders berechnet.

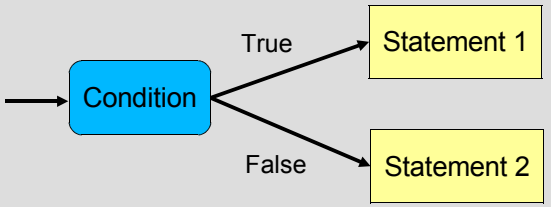
Verzweigungen

- bisher: Anweisungen sequentiell ausgeführt



Verzweigungen

- um auf unterschiedliche Situationen **reagieren** zu können, benötigt man **bedingte** Verzweigungen.



Vergleichsoperatoren

==	a == b	Gleichheit
!=	a != b	Ungleichheit
<	a < b	Kleiner als
>	a > b	Größer als
<=	a <= b	Kleiner gleich
>=	a >= b	Größer gleich

Logische Operatoren

- zur Verknüpfung von Vergleichen

Operator	Beispiel	Erklärung
and	a and b	Wahr, wenn beide wahr sind
or	a or b	Wahr, wenn mindestens eine Variable wahr ist
not	not a	Wahr, wenn a falsch ist.

- Beispiel:

  if (alter>=10) and (alter<=19): print 'Teenager'

- **Unwahr:** False, '', 0, None

Übungsbeispiel

- Angabe:

  a,b,c = 3,0,0

- Bestimmen Sie die Wahrheitswerte:

  a and b

  a or b and not c

  a or (b and c) # Klammer optional

  (a or b) and c # Klammer obligatorisch

  (a or b) and not (a and b)

Verzweigungen

```
x = int(raw_input("Please enter an integer: "))
if x < 0:
    x = 0
    print 'Negative changed to zero'
elif x == 0:
    print 'Zero'
elif x == 1:
    print 'One'
else:
    print 'More'
```

Anmerkung: die zu einer Bedingung zugehörigen Befehle werden als **Block** bezeichnet.

Übungsaufgabe

Ein Programm setzt die folgenden Variablen:

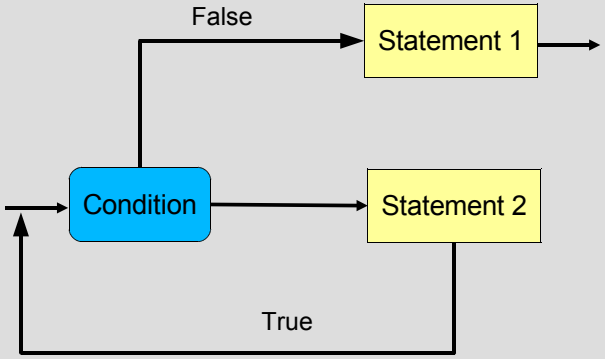
- Name
- Temperatur (in Grad Celsius)

Schreiben Sie Verzweigungen für den folgenden Sachverhalt.

- Temperatur **kleiner als** 20 Grad ist
  => Ausgabe: „Für *Name* ist es hier zu kalt.“
- Temperatur **höher als** 22 Grad
  => Ausgabe: „Für *Name* ist es zu heiß.“
- Liegt die Temperatur im Bereich **von 20 bis 22** Grad
  => Ausgabe:
  „Für *Name* ist die Temperatur von *Temperatur* Grad optimal.“

Schleifen

- ermöglichen es Programmteile zu wiederholen



Die „while“ Schleife

- Die Schleife wird ausgeführt, solange die Bedingung erfüllt ist.

```
counter = 0

while counter < 10:
   print 'Zähle noch ...', counter
   counter +=1


print 'Habe endlich %s erreicht!' % (counter)
```

Übungsaufgabe

- Schreiben Sie eine Programm, das in zweierschritten von 100 bis 80 zählt.
- Der Output soll folgendermaßen aussehen:

  1. Wert: 100

  2. Wert: 98

  3. Wert: 96

  ...

  11. Wert: 80

Übungsaufgabe - Lotto

- Schreiben Sie ein Programm, das 6 ganzzahlige Zufallszahlen zwischen 1 bis 45 ausgibt.
- Ausgabe:

  1. Zufallszahl:

     ...

  6. Zufallszahl:

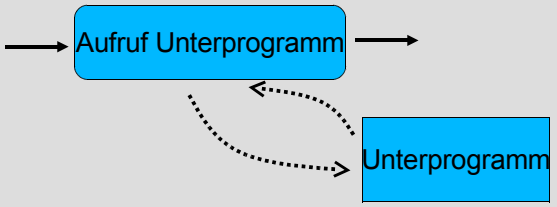
- **Hinweis:** Zufallszahlen werden in python wie folgt generiert:

  import random

  zufallszahl = random.randint(kleinsterWert, höchsterWert)

Funktionen

- Programme enthalten oft immer wiederkehrende Befehlsfolgen. Diese können in Subroutines = Unterprogramme ausgelagert werden.



Funktionen

- Subroutines können auch als mathematische Funktionen aufgefasst werden:

$$\underbrace{y}_{\text{Rückgabewert}} = \underbrace{f(x_1, x_2, \ldots)}_{\text{Argumente}}$$

Funktionen

- Definition von Funktionen:

```
def myFunc(argumente):
    ''' documentation '''
    ...
    return rückgabeWert
```

- Aufruf mittels Funktionsname()

```
myFunc(argumente) bzw. myFunc()
```

- Beispiel:

```
def zylinderVolumen(r,h):
    return r**2*3.14*h
```

Übungsaufgabe

- Schreiben Sie eine Subroutine mit Namen `aktienwert`, die den Kurs und die Anzahl der Aktien übernimmt und den derzeitigen Wert des Portfolios zurückgibt. Rufen Sie die Subroutine aus dem Hauptprogramm auf.

Übungsaufgabe*

- Schreiben Sie ein Programm, das den folgenden Output erzeugt (die erste Zeile hat 20 Zeichen):
- 01010101010101010101
  01010101010101010
  0101010101010101
  010101010101010
  ...
  01
  0
- **Hinweis:** Hier ist es nötig Schleifen zu verschachteln oder Subroutinen zu verwenden.

Kontrollstrukturen in Schleifen

- Zum nächsten Element springen: continue
- Die Schleife abbrechen: break

```
for element in xrange(10):
  if element == 6:
     continue
  elif element == 8:
     print 'Acht ist das letzte Element...'
     break
  print element
```