Informatik I
Einführung in Python, Basics
Vergleich mit C++

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de

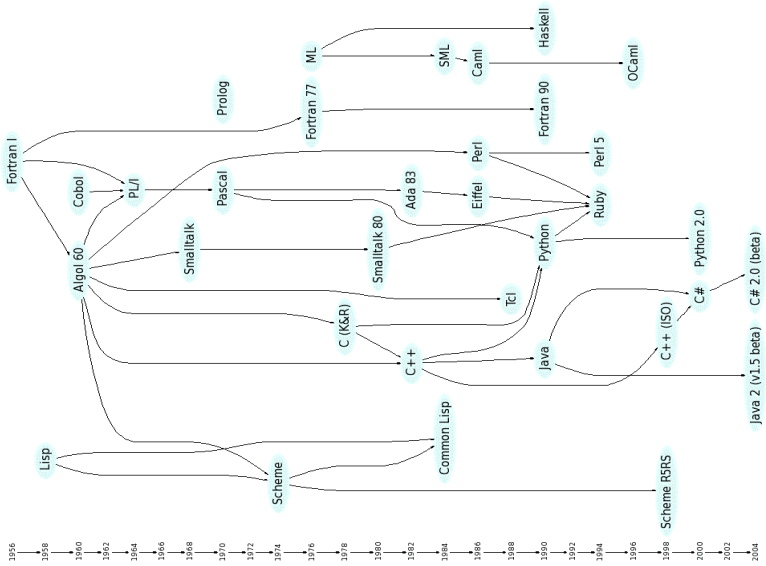
Intro

- Skript-Sprache
  - Nich compiliert, sondern "interpretiert"
  - "Glue"-Sprache (Filter-Skripte, Prototyping, …)
- Erfinder: Guido von Rossum
- Entstehung: 1987
- Web-Seite: www.python.org
- Newsgroup: comp.lang.python
- Sehr aktive Weiterentwicklung (PEPs):
  - "Python Enhancement Proposal"
  - http://www.python.org/peps/
  - Wohldefinierte Prozedur unter Einbeziehung der Community
- Aussprache: wie "Monty Python"

Vergleich mit C++

- Annahme: alle Informatik-I-Höhrer höhren auch "Programmieren"
- Gegenüberstellung von Syntax und Features

Programming Languages History



Warum Python?

- Relativ moderne Sprache
- Leicht erlernbar: wenig Keywords, klare Konzepte
  - "Life's better without braces" (Bruce Eckel)
- Features:
  - dynamisch typisierte Sprache (C++ ist statisch typisiert)
  - Höhere Datenstrukturen (Listen, Dictionaries, …)
  - Introspektion
  - Lambda-Kalkül

→ VHLL (*very high-level language*)

- Erweiterbarkeit:
  - Python kann in C++ eingebettet werden
  - C++ kann von Python aus aufgerufen werden

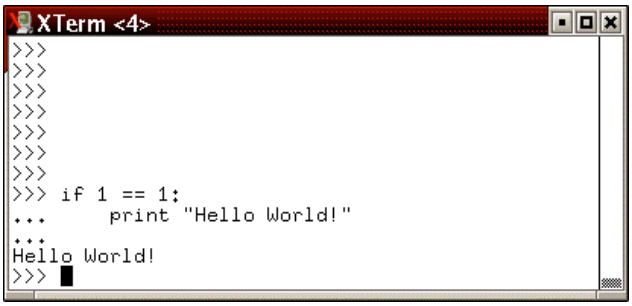
Hello World

- Muß jeder "echte" Programmierer einmal geschrieben haben!

- the ACM "Hello World" project:
  http://www2.latech.edu/~acm/HelloWorld.shtml
- Für erfahrene Programmierer:
  http://www.gnu.org/fun/jokes/helloworld.html
- A propos "Real Programmers":
  http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?Real+Programmers+Don't+Use+Pascal

Interaktive Python-Shell

- **`python`** im Terminal aufrufen
- Kommandozeilen-History
- Kommandozeilen-Editor
- Mehrzeilige Konstrukte mit Leerzeile abschließen
- Beispiel:

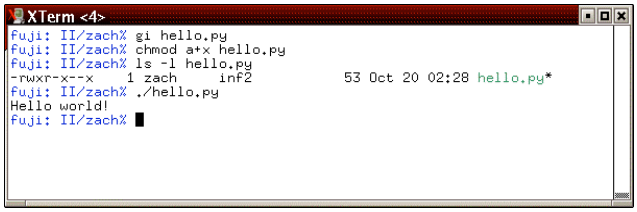


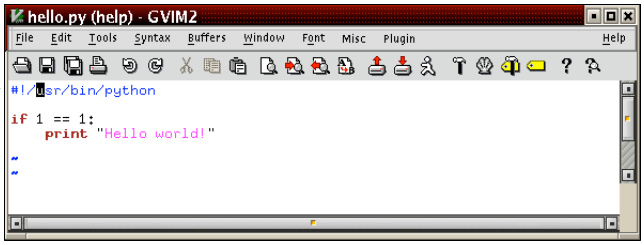
Python IDE's: IDLE und eric

- IDLE: Editor, Console, und Debugger
- Ist bei der Default-Python-Inst. Dabei
- Eric:
  - Sehr schöne Features
  - Brauchen wir nicht (erst sinnvoll bei sehr großen Projekten)
  - Installation sehr aufwendig

Python-Skripte

- Skript = Text-File mit gesetztem exec-Permission-Bit
- Beispiel:





Pfad kann evtl. anders lauten; bekommt man mittels **`which python`**

Woraus besteht ein Programm?

- Abstrakt: Zeichenfolge entsprechend einer formalen Grammatik
- Formale Grammatik besteht aus Regeln folgender Art:

```
statement-list :
    <Leerzeile>                          ["Rek.-Ende"]
    statement <NL> statement-list          [Newline]
```

  - Spätere Vorlesung zu formalen Grammatiken


Kommentare

- Werden ignoriert
- Startet mit #, bis zum Ende der Zeile

```
x = 10  # Bedeutung der Variable
# Kommentarzeile
```

- Zum Vergleich in C++

```
const int Ntries;  // the rest of the line …
// … is treated like a comment
```



- Langer Kommentarblock

```
"""
Blubber
bla bla bla bla.
"""
```

  - Kann überall sort stehen, wo ein Statement stehen kann
  - Funktioniert nicht so allgemein in der interaktiven Pyhton-Shell
- In C/C++

```
const int Ntries;
/*  this is a multiline comment:
    Everything is treated like a comment.
    Comments can't be nested. The comment is
    closed with a */
```

Identifier

- Wie in C++ und praktisch jeder anderen Sprache
- Anderes Wort für Name (Variablename, Funktionsname)
- Zeichenkette
  - Zugelassen: alphanumerische Zeichen und Underscore (_)
  - Erstes Zeichen darf nicht Ziffer sein
    - blub und _bla sind ok
    - 2pi nicht ok
- Kein Limit auf Länge
- Case-sensitiv

Keywords

- Wörter mit spezieller Bedeutung
- Sind reserviert, kann man nicht als Namen verwenden

Keywords in Python				
and	del	for	is	raise
assert	elif	from	lambda	return
break	else	global	not	try
class	except	if	or	while
continue	exec	import	pass	yield
def	finally	in	print	

In C++ :

keyword			
asm	else	operator	throw
auto	enum	private	true
bool	explicit	protected	try
break	extern	public	typedef
case	false	register	typeid
catch	float	reinterpret_	typename
char	for	return	union
class	friend	short	unsigned
const	goto	signed	using
const_cast	if	sizeof	virtual
continue	inline	static	void
default	int	static_cast	volatile
delete	long	struct	wchar_t
do	mutable	switch	while
double	namespace	template	
dynamic_cast	new	this	

Eingebaute (*built-in*) einfache Typen

- **`int`** : Integers (ganze Zahlen)
- **`bool`** : Wahrheitswerte
- **`float`** : Floating-Point-Zahlen (floats) ("reelle Zahlen")
- **`str`** : Strings (im Gegensatz zu C++ echter Datentyp)
- Höhere Datenstrukturen
  - Liste, Komplexe Zahlen, Dictionary: später
  - Set, Module, File, …
- Unterschiede zu C++:
  - **`int`** kann beliebig groß werden (Python schaltet intern autom. um)
  - **`float`** = **`double`** (ex. kein single precision float)
  - Höhere Datenstrukturen (**`str`**, Sequenzen, **`complex`**, **`dict`**, etc.) fest eingebaut, nicht über Header-Files / Libraries

Literale

- Im großen ganzen wie in C++:

Zahlen	
`123 0123 0x123`	**int's (decimal, octal, hex)**
`3.14 3E1`	**floats**
`"" "123"`	**string (first = null string)**
`"name"`	
`"a \"string\""`	**prints: a "string"**
`"""a string ...`	**string over**
`spanning two lines"""`	**several lines is built-in**
`True False`	**bool's**
`None`	**"Zeiger-Wert" / 0-Obj. (NULL in C++)**
`() (1,) (1,2,3)`	**Tupel**
`[] [1,2,3]`	**Liste**
`type(x)`	**Typen kann man vergleichen und zuweisen**

Variablen

- Ähnliche Funktion wie in Mathematik, speichert Wert
- Variable = Paar von Name und "Zeiger" (*Binding*)
- In Python, im Unterschied zu C++:
  - Variablenname hat keinen Typ!! Wert der Variable sehr wohl!
  - Ist nur "Zeiger" auf Wert im Speicher, kann beliebig oft geändert werden
  - Braucht nicht deklariert werden!
  - Muß natürlich vor Verwendung erzeugt worden sein
- Erzeugung / Initialisierung:

```
pi = 3.1415926             # erzeugt Variable
seconds_per_day = 60*60*24 # dito
seconds_per_day = 86400    # ändert Wert
```


- Objekte im Speicher haben immer einen Typ

```
>>> a = 1
>>> type(a)
<type 'int'>
>>> a = "Hello"
>>> type(a)
<type 'string'>
>>> type(1.0)
<type 'float'>
```

Style Guidelines für Variablen

- Wie in C++ und jeder anderen Sprache
- "Kleine" Variablen:
  - Laufvariablen, Variablen mit sehr kurzer Lebensdauer
  - 1-3 Buchstaben
  - `i,j,k,...` für int's
  - `a,b,c,x,y,z` ... für float's
- "Große" Variablen:
  - Längere Lebensdauer, größere Bedeutung
  - Labeling names! ("Sprechende Namen")
  - `mein_alter, meinAlter, determinante, ...`

Operatoren und Ausdrücke

- Ausdruck (*expression*) = mathematischer oder logischer Term
- Beispiele:

```
import math
sin(x)*sin(2*x)      # arithm. expression
"to " + "be"         # string expr.
2 < 1                # logic expression
(x > "aaa") and (x < "zzz")    # dito
```

- Ausdruck setzt sich zusammen aus:
  - Literalen (Konstanten), Variablen, Funktionen
  - Operatoren
  - Klammern
  - Übliche Regeln

Operatoren und Ausdrücke

Arithm. Operator			Meaning
-i	+w		unary + and -
a*b	a/b	i%2	mult., div., modulo
a+b	a-b		binary + and -
x**y			power

In Python im Gegensatz zu C++: Definiert für alle numerischen Typen

Bit-wise Operators	
~ i	bitwise Complement
i & j	bitwise AND
i | j	bitwise OR
i ^ j	bitwise XOR

Relational Operators	
<	less than
>	greater than
<=	less or equal
>=	greater or equal
==	equals
!=	not equal

In Python im Gegensatz zu C++: Definiert für alle Typen!

Assignment op.	equivalent
x □= y	x = x □ (y)
Bsp.:	
x -= y	x = x - y
x /= y	x = x / y
x &= y	x = x & y

Boole'sche Operatoren

- in Python

Boolean Operators	
not	unary not
and	logical and
or	logical or

- Beispiele

```
not x
(not x) and y
((not x) and y) or (z and w))
```

- in C++

Boolean Operators	
!	unary not
&&	logical and
||	logical or

Increment- / Decrement-Operatoren

- In C++:

Self-increment and decrement	Equivalent to
`k = ++j;`	`j=j+1; k=j;`
`k = j++;`	`k=j; j=j+1;`
`k = --j;`	`j=j-1; k=j;`
`k = j--;`	`k=j; j=j-1;`

- Gibt es in Python nicht! ("das ist auch gut so")
- Problem mit Increment- / Decrement-Operatoren:
  - Nebeneffekt → schlechte Wartbarkeit; und
  - Reihenfolge der Auswertung der Operanden nicht festgelegt → versch Ergebnisse
  - Beispiel: Tabelle oben
  - Beispiel 2 & 3:

```
while ( source[j] )
        dest[i++] = source[j++] ;
```

```
i = 2;
j = (i++) * (i--);
j = ?
```


Ganzzahlarithmetik

- Wertebereich
  - Integers können beliebig lang werden
- Overflow:
  - Kein Überlauf in Python!
- Division:
  - wie in C++: 3/2 → 1, 3.0/2 → 1.5
  - Ändert sich wahrschl demnächst (Operator `//`)
- Division und Modulo: `(x/y)*y + x%y == x`

```
import sys
print sys.maxint
2147483647
print sys.maxint * 2
4294967294
```

Float-Arithmetik (praktisch identisch zu C++)

- Implementiert IEEE 754-1985 Standard
- Überlauf ("overflow"):
  - Zahl wird zu groß / zu klein
  - Beispiel: max.float * 2
  - Resultat = $+\infty$ bzw. $-\infty$
- Underflow:
  - Zahlen liegen zu dicht an der 0
  - Resultat = +0.0 bzw. -0.0
- NaN (Not a Number) (in C++):
  - Rechnungen, wo kein sinnvoller Wert rauskommt
  - Bsp.: 1/0 , ∞*0 , sqrt(-1.0)
  - In Python: Fehlermeldung