Praktikum Praktische Informatik

Programmieren in Python
Eine Einführung

Python

Interpreter:

- Programme werden zur Laufzeit von dem sog. Python-Interpreter übersetzt, so dass sie vom Computer verstanden und ausgeführt werden können.
- Im Gegensatz dazu:
  - Compiler: Vor Ausführung werden Programme übersetzt (compiliert).

Python-Programme

Python-Programme besitzen i.d.R. die Endung „**.py**“

Ausführung eines Python-Programms:

```
python <Programm-Name>
```

Alternativen:

- Windows:
  - aus Windows-Oberfläche heraus mit Doppelclick
- Unix:
  - Erste Zeile eine Python-Programmes sollte lauten:
    `#!/usr/bin/env python`
  - Programm ausführbar machen:
    `chmod ugo+x <Programm-Name>`
  - Aufruf:
    `<Programm-Name>`

Python-Shell

Aufruf des Interpreters ohne Programm-Name

- **Interaktiver Modus** wird gestartet:

```
> python
Python 2.0 (#1, Apr 27 2001, 14:34:55)
[GCC 2.95.2 19991024 (release)] on osf1V4
Type "copyright", "credits" or "license" for more
  information.
>>>
```

- Hinter dem “`>>>`“ können Python-Kommandos eingegeben werden
- Eignet sich gut zum Ausprobieren von Kommandos
- Beenden durch
  - `<Ctrl>-D`

Syntaktische Grundlagen

- Ein Kommando pro Zeile
- Mehrere Kommandos pro Zeile durch ";" getrennt
- „Lange Zeilen" können durch ein "\" am Ende der Zeile in der nächsten Textzeile fortgesetzt werden (machts zuweilen übersichtlicher)
- Variablen-Namen:
  - beginnen mit: `A-Za-z_` `(Buchstaben und "_")`
  - dürfen enthalten: `A-Za-z0-9_` `(zusätzlich: Ziffern)`
- Zusammengehörige Anweisungsblöcke (Sequenzen) werden durch **Einrückungen** kenntlich gemacht!

Allgemeine Operatoren

Zuweisungsoperator:

- „="

Vergleichsoperatoren:

- > : größer
- >= : größer oder gleich
- < : kleiner
- <= : kleiner oder gleich
- == : gleich
- != : ungleich (andere Schreibweise: <>)

Boolsche Operatoren:

- and, or, not

Numerische Datentypen (1/2)

Numerische Datentypen in Python:

- Ganzzahl (integer) : kein "." und kein "L"
- Lange Ganzzahl (long) : angehängtes "L" (4L, 15L)
- Gleitkommazahl (float) : mit Dezimalpunkt (4. , 15.) oder Exponentangabe (4e-5)
- Komplexe Zahl : angehängtes " j" (4+3j)

Operationen mit gemischten Datentypen möglich

- Umwandlung in „größeren" Datentyp
- `Ganzzahl < lange Ganzzahl < Gleitkommazahl`

Bem.: bei Ganzzahldivision wird gegen -00 gerundet!

- Bsp.:
  - `1 / 2 -> 0`
  - `(-1)/2 -> -1`
  - `(-1) / (-2) -> 0`

Numerische Datentypen (2/2)

Operation	Result
x + y	sum of x and y
x - y	difference of x and y
x * y	product of x and y
x / y	quotient of x and y
x % y	remainder of x / y
-x	x negated
+x	x unchanged
abs(x)	absolute value or magnitude of x
int(x)	x converted to integer
long(x)	x converted to long integer
float(x)	x converted to floating point
complex(re,im)	a complex number with real part re, imaginary part im. im defaults to zero.
c.conjugate()	conjugate of the complex number c
divmod(x, y)	the pair (x / y, x % y)
pow(x, y)	x to the power y
x ** y	x to the power y

Strings (1/4)

- Zeichenketten stehen in doppelten “Fred“ oder einfachen ‘Foffel‘ Anführungszeichen.
- Zeichenketten können als Folge aufgefasst werden, bei der jedes Zeichen indiziert ist (negative Indizes zählen von rechts):

```
name = "Fred"
name[0] -> "F";  name[3] -> "d";  name[-2] -> "e"
```

- Zeichenketten sind **unveränderbar**!
  - `name[2] = "i" NICHT MÖGLICH!`

Strings (2/4)

Operation	Result
x in s	1 if an item of s is equal to x, else 0
x not in s	0 if an item of s is equal to x, else 1
s + t	the concatenation of s and t
s * n , n * s	n copies of s concatenated
s[i]	i’th item of s, origin 0
s[i:j]	slice of s from i to j
len(s)	length of s
min(s)	smallest item of s
max(s)	largest item of s

Strings (3/4)

Einige String-Methoden:

```
center (width)
      Return centered in a string of length width. Padding is done using spaces.
count (sub[, start[, end]])
      Return the number of occurrences of substring sub in string S[start:end].
      Optional arguments start and end are interpreted as in slice notation.
find (sub[, start[, end]])
      Return the lowest index in the string where substring sub is found, such
      that sub is contained in the range [start, end). Optional arguments start
      and end are interpreted as in slice notation. Return -1 if sub is not found.
replace (old, new[, maxsplit])
      Return a copy of the string with all occurrences of substring old
      replaced by new. If the optional argument maxsplit is given, only the
      first maxsplit occurrences are replaced.
split ([sep[,maxsplit]])
      Return a list of the words in the string, using sep as the delimiter
      string. If maxsplit is given, at most maxsplit splits are done. If sep is
      not specified or None, any whitespace string is a separator.
strip ()
      Return a copy of the string with leading and trailing whitespace removed.
```


Strings (4/4)

String Methoden:

```
capitalize ()
center (width)
count (sub[, start[, end]])
encode ([encoding[,errors]])
endswith (suffix[, start[, end]])
expandtabs ([tabsize])
find (sub[, start[, end]])
index (sub[, start[, end]])
isalnum ()
isalpha ()
isdigit ()
islower ()
isspace ()
istitle ()
isupper ()
join (seq)
ljust (width)
lower ()
lstrip ()
replace (old, new[, maxsplit])
rfind (sub[,start[,end]])
rindex (sub[, start[, end]])
rjust (width)
rstrip ()
split ([sep[,maxsplit]])
splitlines ([keepends])
startswith (prefix[, start[, end]])
strip ()
swapcase ()
title ()
translate (table[, deletechars])
upper ()
```

Listen (1/2)

- Liste von Elementen beliebigen Typs:
  - `liste = [1, 2, "bla", "fasel"]`
- Kennzeichen: Eckige Klammern!
- Zugriffe auf Listenelemente ist per Index möglich:

```
liste[2]    -> ["bla"]
liste[1:3] -> [2, "bla"]
```

- Listen sind **veränderbar**!
  - Zuweisungen an einzelne Elemente
  - Hinzufügen und Löschen von Elementen

```
liste[3] = "blubber"   : aus „fasel" wird „blubber"
liste.append("gargar") : „gargar" hinnedran
liste.remove("bla")    : 1.Auftreten von „bla" löschen
```


Listen (2/2)

Operation	Result
s[i] = x	item i of s is replaced by x
s[i:j] = t	slice of s from i to j is replaced by t
del s[i:j]	same as s[i:j] = []
s.append(x)	same as s[len(s):len(s)] = [x]
s.extend(x)	same as s[len(s):len(s)] = x
s.count(x)	return number of i's for which s[i] == x
s.index(x)	return smallest i such that s[i] == x
s.insert(i, x)	same as s[i:i] = [x] if i >= 0
s.pop([i])	same as x = s[i]; del s[i]; return x
s.remove(x)	same as del s[s.index(x)]
s.reverse()	reverses the items of s in place
s.sort([cmpfunc])	sort the items of s in place

Bem.: Listen-Operationen führen „in-place" Modifikationen durch!

Dictionary (1/2)

- Liste von Elementen beliebigen Typs, wobei jedes Element unter einem Schlüssel beliebigen Typs abgelegt ist.
  - Eintrag von Wertepaaren:
    - `Schlüssel:Wert` (engl.: key:value)
- Kennzeichen: Geschweifte Klammern!

```
telnum = {"martin":28361, "sascha":28359}
```

- Zugriffe auf Elemente ist über die Schlüssel möglich:

```
telnum[„martin"] -> 28361
```

Bem.: Bei Zugriff wird eckige Klammer verwendet!

- Dictionaries sind **veränderbar**!
  - Zuweisungen an einzelne Elemente
  - Hinzufügen und Löschen von Elementen

Dictionary (1/2)

Operation	Result
len(a)	the number of items in a
a[k]	the item of a with key k
a[k] = v	set a[k] to v
del a[k]	remove a[k] from a
a.clear()	remove all items from a
a.copy()	a (shallow) copy of a
a.has_key(k)	1 if a has a key k, else 0
a.items()	a copy of a's list of (key, value) pairs
a.keys()	a copy of a's list of keys
a.update(b)	for k in b.keys(): a[k] = b[k]
a.values()	a copy of a's list of values
a.get(k[, x])	a[k] if a.has_key(k), else x
a.setdefault(k[, x])	a[k] if a.has_key(k), else x (also setting it)

Selektion

Bedingte Programmausführung:

```
if x < 0:
    print x, "ist negativ"
elif x == 0:
    print x, "ist Null"
else:
    print x, "ist positiv"
```

- `elif` ist eine Abkürzung für `else if`
- es sind beliebig viele `elif`-Klauseln möglich
- `else`-Klausel ist optional

Iteration (1/2)

Zählschleife `for`:

```
for i in ["Anton", "Berta", "Egon"]:
    print i[0], " wie ", i
```

- `for` iteriert über alle Elemente einer Liste
- Bem.: Die Folge, über die iteriert wird, sollte man im Inneren der Schleife nicht ändern.
- Für „richtige“ Zählschleife braucht man „Zählliste“:
  - [0,1,2,3,..]
  - erhält man durch spezielle Funktion: `range()`
  - Bsp.: `range(5) -> [0,1,2,3,4]`

Iteration (2/2)

Schleife mit Vorbedingung `while`:

```
a = 0; b = 1
while b < 10:
    print b
    a, b = b, a + b
```

- `while`-Schleife wird ausgeführt, solange Bedingung wahr ist
- alles außer 0 ist „wahr“, 0 ist „falsch“

Funktionen

- Funktionsdefinitionen werden durch das Schlüsselwort `def` eingeleitet
- Funktionsrumpf wird durch Einrückung gekennzeichnet
- Funktionen:
  - können Übergabeparameter besitzen
  - können Rückgabeparameter besitzen (return)
- Bsp.:

```
# Fibonacci-Zahlen bis n berechnen und ausgeben
def fib(n):
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a + b
```

Module

- Für Python existiert eine Reihe von Modulen, die Funktionen bereitstellen
- mit `import` werden Module eingebunden
- Bsp.:

```
import math
math.sin(math.pi)
```

- Zugriff auf Funktionen des Moduls über Angabe des Modulnames, gefolgt von „.“, gefolgt von Funktionsname!
- Man greift auf Namensraum des Moduls zu
- Standardmodule: math, string, rand

Dokumentation

`http://www.tm.informatik.uni-frankfurt.de/python`

Originaldokumentation:

- Tutorial
- Reference Manual
- Library Reference