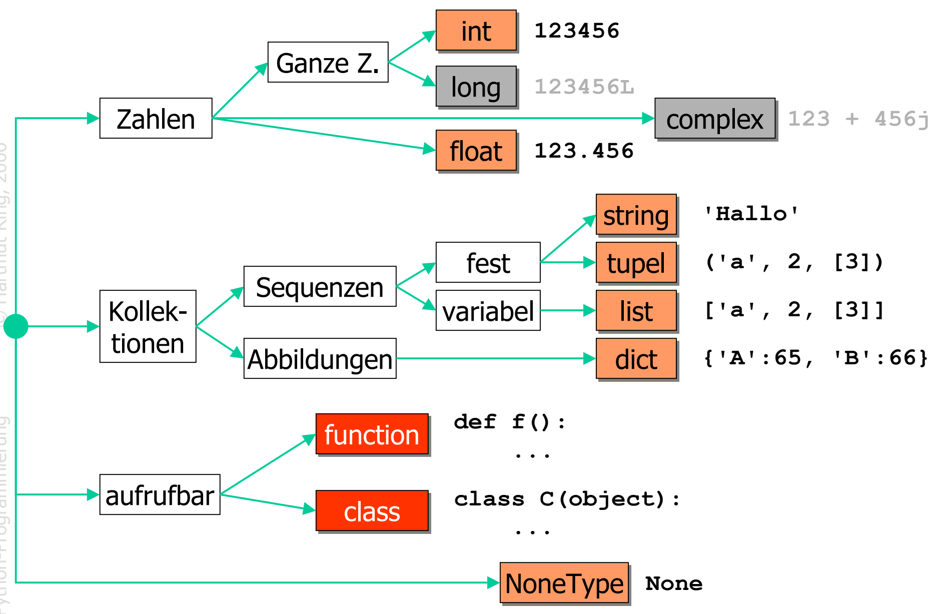
Eingebaute Python-Datentypen



Datentyp `string`

`s = 'Das ist ein String'`	
`s = ''`	Leerer String
`len(s)`	Länge

`s2 = """String über`
`mehrere Zeilen"""`	

`s = s1 + s2`
`s += s2`
	Verkettung

`s[i]`	Indizierung
`s[i:k]`	Teilstring ("Slice")
`s[i:]`	
`s[:k]`	
`s[i:-1]`	
`if 'x' in s:`	Element-Beziehung
`for c in s:`	Iteration

string-Methoden	
capitalize	ljust
center	lower
count	lstrip
decode	replace
encode	rfind
endswith	rindex
expandtabs	rjust
find	rstrip
index	split
isalnum	splitlines
isalpha	startswith
isdigit	strip
islower	swapcase
isspace	title
istitle	translate
isupper	upper
join	zfill

```
''.dir()
```

Idiome mit Python-Strings

Wahlweise einfache oder doppelte **Anführungszeichen**	'Wer hat die "Buddenbrooks" geschrieben?' "don't worry"
Escape-Sequenzen mit \	'erste Zeile**\n**zweite Zeile" 'Text mit **\\'**einfachen**\\'** und "doppelten" Anführungszeichen' "Text mit 'einfachen' und **\\"**doppelten**\\"** Anführungszeichen" 'c:\\\\Daten\\\\neu'
raw strings (keine Escape-Sequenzen)	**r**'c:\Daten\neu'
Textwiederholung mit Multiplikationsoperator	'Ro' + 2 * 'ko' → 'Rokoko'
%-Zeichen: **Platzhalter** %s für String %d für Dezimalzahl	'Sommer **%d**' **%** 2006 → 'Sommer 2006' '**%02d**. **%s** **%d**' % (1, 'Mai', 2006) → '01. Mai 2006'

Sequenzen string, tuple, list

Konstruktion	
a = 'abc'	String
a = ()	Leeres Tupel
a = (1, 'x', [3,4])	polymorphes Tuple
a = []	Leere Liste
a = [1, 'x', [3,4]]	polymorphe Liste

Operatoren	
len(a)	Länge
a = a1 + a2	Verkettung
a += a2	
a = 3 * a1	Vervielfachung
a < b etc.	Lexikograph. Vergleich
if 'x' in a:	Element-Beziehg.
for c in a:	Iteration

Indizierung, Slices	
a[i]	Indizierung
a[i] = 3	
a[i:k], a[i:], a[:k], a[i:-1]	Teilliste ("Slice")
a[i:k] = a1	Ersetzung einer Teilliste
del a[i]	Element löschen
del a[i:k]	Teilliste löschen

$$\begin{tikzcd}
 & & \text{string} \\
 & \text{fest} \arrow[ur] \arrow[r] & \text{tupel} \\
\text{Sequenzen} \arrow[ur] \arrow[r] & \text{variabel} \arrow[r] & \text{list}
\end{tikzcd}$$

gemeinsame Methoden für string und list

```
count(x)
index(x)
```

string-Methoden	
capitalize	ljust
center	lower
count	lstrip
decode	replace
encode	rfind
endswith	rindex
expandtabs	rjust
find	rstrip
index	split
isalnum	splitlines
isalpha	startswith
isdigit	strip
islower	swapcase
isspace	title
istitle	translate
isupper	upper
join	zfill

list-Methoden

```
append(x)
extend(l)
insert(i,x)
pop([i])
remove(x)
reverse()
sort([f])
```

"List Comprehensions"

$\{2i \mid i \in \{1,2,3\}\}$

`[ 2*i  for  i in [1,2,3] ]`   [2, 4, 6]

`[ x.upper() for x in 'abcdef' ]`   ['A', 'B', 'C', 'D', 'E', 'F']

$\{10n \mid n \in \{0,\ldots,9\}, n \bmod 5 = 0\}$ [10, 20, 30, 40, 60, 70, 80, 90]

`[10*n for n in range(10) if n % 5 != 0]`

Übungsaufgaben:

`[3*i for i in range(100) if (3*i) % 10 == 3]`   vereinfachen !

`[str(round(math.pi, n)) for n in range(1,10)]`

`''.join (['NAS'[int(i==0) : 2+int(i==2)] for i in range(3)])`

Dictionaries

$$\begin{tikzcd}
 & & & \text{string} \\
 & & \text{fest} \arrow[ur] \arrow[r] & \text{tupel} \\
 & \text{Sequenzen} \arrow[ur] \arrow[dr] & & \\
\text{Kollektionen} \arrow[ur] \arrow[dr] & & \text{variabel} \arrow[r] & \text{list} \\
 & \text{Abbildungen} \arrow[rr] & & \text{dict}
\end{tikzcd}$$

Konstruktion

```
d = {1: 'eins', 'x': (1,2,3)}
d = dict(x=17, y=3)    #  d = {'x':17, 'y':3}
```

Operatoren	
len(d)	Anzahl der Elemente
if 'x' in d:	Element-Beziehg.
for c in d:	Iteration

Indizierung	
d[x]	Indizierung
d[x] = y	
del d[x]	Element löschen

wichtige dict-Methoden

```
copy()
has_key(k)
items()
keys()
values()
get(k[,x])
clear()
update(d)
```

Python-Schlüsselwörter

Junktoren	Funktionen, Klassen	Import	geplant
and	def	from	True
or	global	import	False
not	return		as
	yield		
	pass		
	class		

strukturierte Anweisungen	Ausnahmen	Sonstige
if	try	print*
elif	except	assert
else	finally	in
while	raise	is
for		del
break		exec*
continue		lambda*

* Abschaffung geplant!

Operatoren

+	+=		<
-	-=		>
*	*=		<=
/	/=	Gleitkommadivision	>=
//	//=	Ganzzahldivision	==
%	%=		!=
**	**=	Potenz	<>
&	&=		
|	|=		
^	^=		
>>	>>=		
<<	<<=		
~			

Python 2.2 bis 2.4:

```
from __future__ import division
7 / 3
7 // 3
```

Funktionen

Feste Parameterliste

```
def fillString(text, fill):
    """ fill zwischen je zwei Zeichen
        von text setzen """
    return fill.join(list(text))
```

```
fillString('Mntkl', 'e')
```

Benannte Parameter

```
fillString(fill='e', text='Mntkl')
```

Optionale Parameter

```
def f1(s, n=2):
    """ String s n mal hintereinander """
    return n * s
```

```
f1('la', 3)           lalala
f1('la')              lala
f1(n=4, s='la')       lalalala
f1(s='la')            lala
f1(n=4)               Fehler (s fehlt)
```

Beliebig viele Parameter

```
def f2(a, *opt):
    """ Produkt beliebig vieler Faktoren """
    for x in opt:
        a *= x
    return a
```

Schlüsselwort-Parameter

```
def f3(a, **d):
    """ nach a können beliebig viele weitere Parameter kommen.
        Diese werden als Dictionary übergeben. """
    print 'a = %s' % str(a)
    for var in d:
        print '%s = %s' % (var, str(d[var]))
```

Kombination

```
def f4(a, b=1, *opt, **d):
    print 'a = %s, b = %s' % (str(a), str(b))
    for x in opt:
        print 'optionaler Parameter = %s' % str(x)
    for var in d:
        print '%s = %s' % (var, str(d[var]))
```

Eingebaute Funktionen (1/3)

Konvertierungen

dict([mapping-or-sequence])
tuple([sequence])
list([sequence])
str(object)
chr(i)
int(x[, radix])
long(x[,radix])
float(x)
bool(x)
ord(c)
complex(real[, imag])
hex(x)
oct(x)
coerce(x, y)

Kollektionen

len(s)
range([start,] stop[, step])
xrange([start,] stop[, step])
slice([start,] stop[, step])

Mathematik

abs(x)	auch für komplexe Zahlen
max(s[, args...])	
min(s[, args...])	
round(x,n)	n Stellen
cmp(x,y)	= sgn(x-y)
divmod(a, b)	
pow(x, y[, z])	= x^y mod z

Eingebaute Funktionen (2/3)

Funktionale Programmierung
apply(function, args[, keywords])
eval(expression[, globals[, locals]])
filter(function, list)
map(function, list, ...)
reduce(function, sequence[, initializer])
zip(seq1, ...)

IO
file(filename[, mode[, bufsize]])
Synonym:
open(filename[, mode[, bufsize]])
input([prompt])
raw_input([prompt])

Introspektion
callable(object)
buffer(object[, offset[, size]])
dir([object])
globals()
locals()
help([object])
isinstance(object, classinfo)
issubclass(class1, class2)
type(object)
vars([object])

Eingebaute Funktionen (3/3)

System	
compile(string, filename, kind[, flags[, dont_inherit]])	
execfile(file[, globals[, locals]])	
hash(object)	Wert
id(object)	Objekt
intern(string)	
reload(module)	
repr(object)	

Unicode
unichr(i)
unicode(object[, encoding[, errors]])

Klassen
hasattr(object, name)
getattr(object, name[, default])
setattr(object, name, value)
delattr(object, name)

Iteration
iter(o[, sentinel])

undokumentiert	
exit	copyright
quit	credits
object	license
property	
staticmethod	
classmethod	
super	

Klassen

Prozedurorientiert

```
def rectInit (x, y, w, h):
    return [x, y, w, h]

def rectArea(r):
        return r[2] * r[3]

def rectOffset(r, dx, dy):
        r[0] += dx
        r[1] += dy
```

```
r = rectInit (0,0, 10,20)

rectOffset (r, 10,20)
print rectArea (r)
```

Konstruktor

prozedur (objekt, parameter)

Objektorientiert

```
class Rectangle (object):
   def __init__(self, x, y, w, h):
       self.x = x
       self.y = y
       self.w = w
       self.h = h

   def area (self):
       return self.w * self.h

   def offset (self, dx, dy):
       self.x += dx
       self.y += dy
```

Konstruktor

Methoden

```
r = Rectangle (0,0, 10,20)

r.offset (10,20)
print r.area()
```

objekt.**methode** (parameter)

Vergleich Java – Python

Java

```
class Rectangle extends Object {
   public Rectangle ( double x, double y,
                      double w, double h) {
      this.x = x;
      this.y = y;
      this.w = w;
      this.h = h;
   }

   double area() {
      return w * h;
   }

   void offset (double dx, double dy) {
      x += dx;
      y += dy;
   }

   double x;
   double y;
   double w;
   double h;
}
```

```
public class JavaVergleich {
  public static void main (String[] args){
    Rectangle r = new Rectangle (0,0, 10,20);
    r.offset (10,20);
    System.out.println(r.area());
  }
}
```

Python

```
class Rectangle (object):
   def __init__ (self, x, y, w, h):

      self.x = x
      self.y = y
      self.w = w
      self.h = h


   def area (self):
      return self.w * self.h


   def offset (self, dx, dy):
      self.x += dx
      self.y += dy
```

```
r = Rectangle (0,0, 10,20)
r.offset (10,20)
print r.area()
```