Vererbung

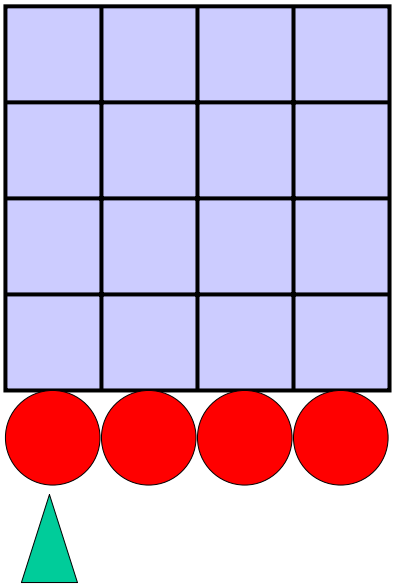

```
class Figure (object):    # Basisklasse
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def area(self): # abstrakte Methode!
        pass

    def offset(self, dx, dy):
        self._x += dx
        self._y += dy
```

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

    def area(self):
        return self._w * self._h
```

```
class Circle (Figure):
    def __init__(self, x, y, r):
        Figure.__init__(self, x, y)
        self._r = r


    def area(self):
        return math.pi * self._r * self._r
```

```
f = [Circle(0,0, 10), Rectangle(0,0, 10,20)]
a = 0
for x in f:
    x.offset(10,20)
    a += x.area()
print a
```


Beispiel für Polymorphie: Backtracking



Backtracking-Basisklasse


```
class BacktrackProblem (object):
    def __init__(self, n):
        self.nLevels = n

    def findSolutions(self):
        self.solutions = []
        curLevel = 0
        self.prepare()
        self.startObject (curLevel);
        while curLevel >= 0:
            moved = self.move(curLevel)
            while moved and not self.valid(curLevel):
                moved = self.move(curLevel)
            if moved: # erlaubte Stellung gefunden
                if curLevel == self.nLevels-1: # Lösung!
                    self.solutions.append(self.repr())
                else:
                    curLevel += 1
                    self.startObject(curLevel)
            else:
                self.removeObject(curLevel)
                curLevel -=1
```

Eine konkrete Klasse muss die folgenden abstrakten Methoden implementieren:

```
def prepare(self):
    pass

def move(self, n):
    pass

def valid(self, n):
    pass

def startObject(self, n):
    pass

def removeObject(self, n):
    pass

def repr(self):
    pass
```

Vorbereitung für jeden Lösungsversuch (nur falls nötig).

Für n-tes Vektorelement den nächsten möglichen Wert einsetzen, unabhängig davon, ob dieser Wert konsistent ist. Rückgabewert 0 <==> kein neuer Wert mehr möglich.

Prüfen, ob das n-te Vektorelement erlaubt ist, unter der Voraussetzung, dass die Vektorelemente 0 ... n-1 erlaubt sind.

n-tes Vektorelement auf einen Pseudowert setzen, so dass es mit move auf den ersten echten Wert gesetzt wird.

n-tes Vektorelement entfernen

aktuelle Stellung (Lösung)

Arithmetische Operatoren

+	**__add__**(*self, other*)	**__radd__**	**__iadd__**
-	**__sub__**(*self, other*)	**__rsub__**	**__isub__**
*	**__mul__**(*self, other*)	**__rmul__**	**__imul__**
/	**__div__**(*self, other*)	**__rdiv__**	**__idiv__**
/	**__truediv__**(*self, other*)	**__rtruediv__**	**__itruediv__**
//	**__floordiv__**(*self, other*)	**__rfloordiv__**	**__ifloordiv__**
%	**__mod__**(*self, other*)	**__rmod__**	**__imod__**
divmod	**__divmod__**(*self, other*)	**__rdivmod__**	
pow, **	**__pow__**(*self, other*[, *modulo*])	**__rpow__**	**__ipow__**
<<	**__lshift__**(*self, other*)	**__rlshift__**	**__ilshift__**
>>	**__rshift__**(*self, other*)	**__rrshift__**	**__irshift__**
&	**__and__**(*self, other*)	**__rand__**	**__iand__**
^	**__xor__**(*self, other*)	**__rxor__**	**__ixor__**
|	**__or__**(*self, other*)	**__ror__**	**__ior__**

```
class Rational (object):
    def __init__(self, p, q=1): # ...
    def __add__ (a, b):
        if not isinstance(b, Rational):
            b = Rational(b)
        return Rational(a.p*b.q - b.p*a.q, a.q*b.q)
    def __radd__ (b, a):
        return b + a
```

```
r = Rational (7,3)
s = r - 1
t = 1 - r
```

r.__sub__(1)

r.__rsub__(1)

Arithmetische Operatoren


```
def gcd(a,b):
    """Größter gemeinsamer Teiler
       (Euklidischer Algorithmus)"""
    assert a >= 0 and b >= 0
    while b > 0:
        a, b = b, a % b
    return a

class Rational (object):
    """Rationale Zahlen, immer in gekürzter Form
mit positivem Nenner"""
    def __init__(self, p, q=1):
        if type(p) == str:   #  z.B. '3/17'
            l = p.split('/')
            p = int(l[0])
            if len(l) > 1:
                q = int(l[1])
        if q < 0:
            p,q = -p, -q  # Nenner immer positiv
        d = gcd(abs(p), q)
        if d > 1:         # kürzen, falls möglich
            p //= d
            q //= d
        self.p = p
        self.q = q
```

```
    def __repr__(self):
        if self.q == 1:
            return str(self.p)
        else:
            return '%d/%d' % (self.p, self.q)

    def __neg__(self):
        return Rational(-self.p, self.q)

    def __add__(a, b):
        if not isinstance(b, Rational):
            b = Rational(b)
        return Rational(a.p*b.q + b.p*a.q, a.q*b.q)

    def __radd__(b, a):
        return b + a

    def __sub__(a, b):
        if not isinstance(b, Rational):
            b = Rational(b)
        return Rational(a.p*b.q - b.p*a.q, a.q*b.q)
    def __rsub__(b, a):
        return -b - -a
```

```
r = Rational(2,3)
print r-1
print 1-r
```


Beispiel: Backup (1)


```
import zipfile, os, time
backupSources = [r'c:\Daten', r'c:\Uni'] # zu sichernde Verzeichnisse
backupFile    = r'c:\backup\bak001.zip'  # Ziel der Sicherung
skip          = ['.bak', '.obj', '.pyc'] # Ausschluss-Endungen
bkTime        = (2006, 05, 31, 23, 59)   # ab Jahr, Monat, Tag, Stunde, Minute

class Backup (object):
    def __init__(self, sources, target, skip, bkTime):
        self.sources, self.target, self.skip = sources, target, skip
        self.bkTime = bkTime
        self.t = time.mktime(self.bkTime+(0,0,0,0))
        self.zip = zipfile.ZipFile(backupFile, 'w', zipfile.ZIP_DEFLATED)
        for path in self.sources:
            self.__backupDir(path)
        self.zip.close()

    def __backupDir(self, path):
        if os.path.isdir(path): # Unterverzeichnis rekursiv sichern
            for relName in os.listdir(path):
                self.__backupDir(os.path.join(path, relName))
        else: # Datei
            for s in self.skip:
                if path.endswith(s): return            # Ausschluss-Erweiterung
            if os.stat(path).st_mtime < self.t: return # alte Datei
            sp = path.replace(':',';')
            try:     self.zip.write (path, sp)
            except:  print path, sp

Backup(backupSources, backupFile, skip, bkTime)
```

Beispiel: **Backup (2)**


```
class ZipFileX (zipfile.ZipFile):
  def write(self, filename, arcname,
            compress_type=zipfile.ZIP_DEFLATED):
    """ zip.write() betrachtet Dateinamen (mit Umlauten)
        als OEM (DOS-Code, Codepage 850) und wandelt sie nach
        ANSI um (Latin-1) um.
        Deshalb müssen unter Windows ANSI-Namen
        zunächst nach OEM uebersetzt werden:
    """
    zipfile.ZipFile.write(self, filename,
                          arcname.decode('latin-1').encode('cp850'),
                          compress_type)
```

```
...
class Backup:
  def __init__(self, sources, target, skip, bkTime):
    ...
    self.zip = ZipFileX(backupFile, 'w')
    ...
...
```


Properties


```
class Temperature (object):
    def __init__(self):
        self.celsius = 0.0

    def getFahrenheit(self):
        return 32.0 + 1.8 * self.celsius
    def setFahrenheit(self, f):
        self.celsius = (f - 32.0) / 1.8

t = Temperature()
t.celsius = 25
print t.getFahrenheit()
t.setFahrenheit(100)
print t.celsius
```

```
class Temperature (object):
    def __init__(self):
        self.celsius = 0.0

    def __getFahrenheit(self):
        return 32.0 + 1.8 * self.celsius
    def __setFahrenheit(self, f):
        self.celsius = (f - 32.0) / 1.8

    fahrenheit = property(
        fget = __getFahrenheit,
        fset = __setFahrenheit)

t = Temperature()
t.celsius = 25
print t.fahrenheit
t.fahrenheit = 100
print t.celsius
```

Iteratoren (1)

für **`for`**-Schleifen über nicht indizierbare Objekte

Ein **Iterator** ist ein Objekt mit einer Methode **`next()`**

Die Methode **`next()`** liefert das nächste Objekt oder eine Ausnahme vom Typ **`StopIteration`**

`iter(obj)` liefert einen neuen Iterator für **`obj`**
ruft **`obj.__iter__()`** auf

Kurzschreibweise:

```
for x in obj:
    f(x)
```

```
t = iter(obj)
while True:
    try:
        x = t.next()
    except StopIteration:
        break
    f(x)
```


Iteratoren (2)

Beispiel: Collatz-Zahlen

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

```
class Collatz:
    def __init__(self, n):
        self.n = n
    def __iter__(self):
        return self
    def next(self):
        if self.n == 1:
            raise StopIteration()
        else:
            result = self.n
            if self.n % 2 == 0:
                self.n = self.n / 2
            else:
                self.n = 3 * self.n + 1
            return result

c = Collatz(7)

for i in c: print i,
```

Generatoren (1)

neues Schlüsselwort **`yield`**

Rückgabe eines Funktionswerts ohne Beenden der Funktion

Python 2.2 `from __future__ import generators`

Python 2.3 *Standard*

```
def genCollatz(n):
    while n > 1:
        yield n
        if n % 2 == 0:
            n = n / 2
        else:
            n = 3*n + 1

g = genCollatz(7)

for i in g: print i,
```

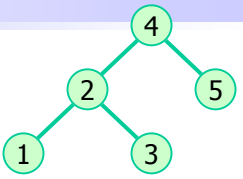
Eine Funktion, in der **`yield`**-Anweisungen vorkommen, heißt **Generatorfunktion**

Der Aufruf einer einer Generatorfunktion liefert ein **Generatorobjekt**

Generatorobjekte sind über die **Iterator**-Schnittstelle ansprechbar

Generatoren (2)

Beispiel: Inorder-Durchlauf in Binärbaum




```
def inorder(t):
    if type(t) == tuple:
        for x in inorder(t[1]):
            yield x
        yield t[0]
        for x in inorder(t[2]):
            yield x
    else:
        yield t


t = (4, (2, 1, 3), 5)

g = inorder(t)

for i in g: print i,
```

Wert

linker Unterbaum

rechter Unterbaum

Generator-Ausdrücke


```
[x*x for x in range(5)]
```

List Comprehension: [0,1,2,3,4]

```
sum([x*x for x in range(5)])
```

sum([0,1,2,3,4])

```
x*x for x in range(5)
```

Generator-Ausdruck

```
g = (x*x for x in range(5))
for i in g:
    print i
```

Iteration über Generator-Ausdruck

```
sum(x*x for x in range(5))
```

```
sum(genCollatz(5))
```

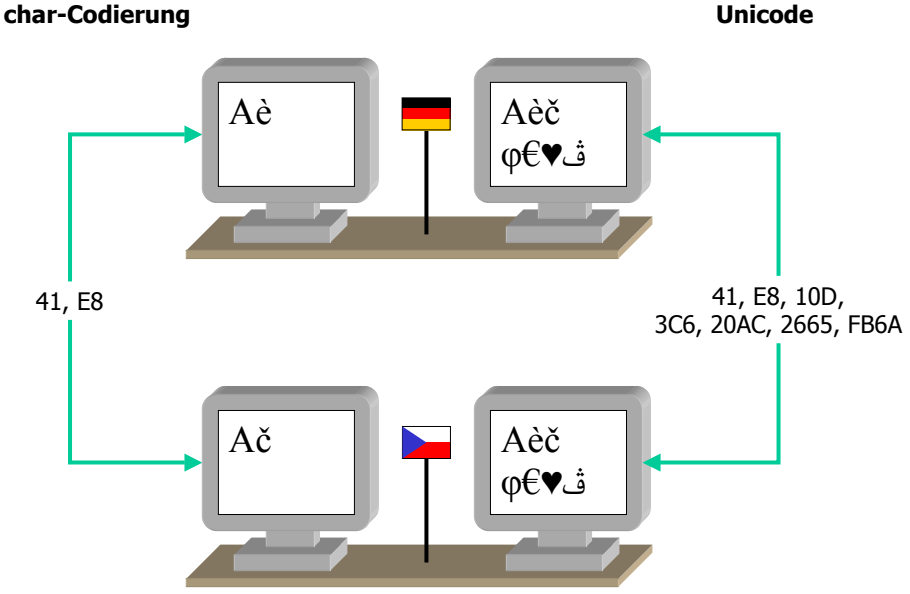
```
sum(x*x for x in genCollatz(5))
```

Summe über Generator-Ausdruck

```
def prod(l):
    p = 1
    for x in l: p *= x
    return p
prod(genCollatz(5))
```

Produkt über Generator-Ausdruck

Unicode



Unicode

- Standard zur Darstellung aller Schriften der Welt und vieler Sonderzeichen (`www.unicode.org`)
- Aktuelle Version: 4.0
- Weit mehr als 2^{16} definierte Zeichen (code points)
- Kodierungen
  - *Für alle code points, z. B.*
    UTF-8, UTF-16.
  - *Für Teilmengen z. B.*
    US-ASCII Teilmenge von UTF-8
    ISO-8859-1 erste 256 code points
    ISO-8859-2, usw.

Unicode: Kodierungen für Teilmengen

Kodierungen für Teilmengen

			ISO-8859-1 (Latin-1)		ISO-8859-2 (Latin-2)	
20	32	→	32		32	
21	33	→	33	!	33	!
…	…					
40	64	→	64	@	64	@
41	65	→	65	A	65	A
…	…					
C4	196	→	196	Ä	196	Ä
…	…					
C8	200	→	200	É	268	Č
…	…					

Unicode: UTF-8

uncodiert (code points)		UTF-8
0..7F (7 bit)	xxxxxxx	0 xxxxxxx
80..7FF (11 bit)	xxxxxxxxxxx	110 xxxxx 10 xxxxxx
800..FFFF (16 bit)	xxxxxxxxxxxxxxx	1110 xxxx 10 xxxxxx 10 xxxxxx
10000..1FFFFF (21 bit)	xxxxxxxxxxxxxxxxxxxxx	11110 xxx 10 xxxxxx 10 xxxxxx 10 xxxxxx

0xxxxxxx	1-Byte-Zeichen (ASCII)
10xxxxxx	Folgebyte
110xxxxx	1. Byte von 2
1110xxxx	1. Byte von 3
11110xxx	1. Byte von 4

Einfache Synchronisation: Ab einer beliebigen Position nach spätestens 3 Bytes.

Aufgabe:
Bis zu welcher Obergrenze ließe sich UTF-8 theoretisch erweitern?

```
if cp <= 0x7F:  # Berechnungsprinzip
    nBytes = 1
    u1 = cp
elif cp <= 0x07FF:
    nBytes = 2
    u2 = 0x80 | (cp & 0x3F);   cp >>= 6
    u1 = 0xC0 | cp
elif cp <= 0xFFFF:
    nBytes = 3
    u3 = 0x80 | (cp & 0x3F);   cp >>= 6
    u2 = 0x80 | (cp & 0x3F);   cp >>= 6
    u1 = 0xE0 | cp
elif cp <= 0x1FFFFF:
    nBytes = 4
    u4 = 0x80 | (cp & 0x3F);   cp >>= 6
    u3 = 0x80 | (cp & 0x3F);   cp >>= 6
    u2 = 0x80 | (cp & 0x3F);   cp >>= 6
    u1 = 0xF0 | cp
```


Unicode: UTF-8

0xxxxxxx	1-Byte-Zeichen (ASCII)
10xxxxxx	Folgebyte
110xxxxx	1. Byte von 2
1110xxxx	1. Byte von 3
11110xxx	1. Byte von 4

Beispiel: Euro-Symbol (€)

Code Point 8364 = 0x20AC

8364 = 00100000 10101100

6-Bit-Gruppierung: 0010 000010 101100

1110 0010	10 000010	10 101100
1. Byte von 3	Folgebyte	Folgebyte
E2	82	AC

als ISO-8859-1 gelesen:

â ‚ ¬

Unicode: UTF-16

uncodiert (code points) **UTF-16**

`0000..D7FF und E000 ..FFFF`

`00000000 00000000 mnnmlkji hgfedcba`

`mnnmlkjihgfedcba`

`(1101 1xxx xxxxxxxx = D800 ... DFFF`
sind reserviert für UTF-16-Codierung!)

// Berechnungsprinzip:

assert cp <= 0xD7FF or cp > 0xFFFF

if cp <= 0xD7FF:
    nWords = 1
    u1 = cp

`0100..1FFFFF` (Ab Unicode 3.1)

`00000000 000utsrq ponmlkji hgfedcba`

`110110utsrqponmlk` `110111jihgfedcba`

else:
    nWords = 2
    u2 = 0xDC | (cp & 0x3FF)
    u1 = 0xD8 | (cp >> 10)

Vorspann:

UTF-16BE	`FEFF`	þÿ
UTF-16LE	`FFFE`	ÿþ

Unicode in Python

	Wert	mit **print**
`u = unicode('ä', 'Latin-1')`	`u'\xe4'`	`ä`
`e = u.encode('Latin-1')` `e.decode('Latin-1')`	`'\xe4'` `u'\xe4'`	`ä` `ä`
`e = u.encode('maclatin2')` `e.decode('Latin-1')`	`'\x8a'` `u'\x8a'`	`Š` □
`e = u.encode('UTF-8')` `e.decode('Latin-1')`	`'\xc3\xa4'` `u'\xc3\xa4'`	`Ã¤` `Ã¤`