Python für Lisper

P. Herth

Python

A user-friendly programming language!

Eigenschaften

- Dynamisch
- REPL
- Standard durch Referenzimplementation
- Python kompiliert zu Bytecode, VM portables C (CPython)
- Alternativ: Jython

Struktur durch Einrückung

```
def signum(x):
    if x > 1:
        return 1
    elif i == 0:
        return 0
    else:
        return -1
```

- Blöcke werden durch Einrückung mit gleicher Tiefe gebildet.
- Keine Klammern `{ }`, kein `begin` oder `end`.

Datentypen

- Alle Typen sind Objekte: `"asdf".upper()` $\longrightarrow$ `"ASDF"`
- Ganzzahlen (beliebig gross), Flieskommazahlen
- Listen (Arrays)
- Strings als immutable Listen
- Tupel (immutable Arrays)
- Hashtables
- Objekte

Beispiele

```
def cons(x,y):
    return (x,y)

def car(list):
    return list[0]

def cdr(list):
    return list[1:]

>>> print cons(4,5)
(4, 5)
>>> print car("asdf"),cdr("asdf")
a sdf
ages = { "Paul" : 27, "Mary" : 25}
ages["Fritz"] = 30
```

Funktionen

```
def add(x):
    y = x + x
    return y
>>> add(2)
4
>>> add("xyz")
'xyzxyz'
```

- Lokale Variablen werden durch Zuweisung definiert
- First-Class Funktionen
- Python ist ein “Lisp-1”

Funktionen

```
def fun(name, title="Mr."):
    print "Hello %s %s." % (title, name)

>>> fun("Miller")
Hello Mr. Miller.
>>> fun(title="Mrs.", name= "Smith")
Hello Mrs. Smith.
```

- Alle Argumente automatisch “Keywords”
- Durch Defaultwerte wird ein Argument optional.

for

```
for i in range(10):
    print i

for i in (1,2,3,5,7,11):
    print 2*i,

for letter in "abcdefghijklmnopqrstuvwxyz":
    print letter,

for line in file("/etc/passwd"):
    fields = line.split(":")
    print "username:",fields[0],"\tshell:",fields[5]
```

- `for` iteriert über Sequenzen
- Sequenzen sind Listen (Arrays) oder beliebigen Iteratoren

Generators

```
def squares(upto):
    i=0
    while i<upto:
        i = i + 1
        yield i*i

for i in squares(10):
    print i,
```

```
1 4 9 16 25 36 49 64 81 100
```

Klassen

```
class Page:
    size = 'a4'                 # allocation class
    def __init__(self):         # Konstruktor
        self.number = 1         # allocation instance

    def print_number(self):
        print self.number

    def __add__(self, other):
        self.number = self.number + other
        return self
```

Klassen

```
>>> p1 = Page()
>>> p2 = Page()
>>> p1 = p1 + 7
>>> p1.number
8
>>> p1.text='asdf'
>>> p1.text
'asdf'
>>> p2.text
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
AttributeError: Page instance has no attribute 'text'
```

CPython

- In C geschriebener Bytecodeinterpreter
- Standardbibliothek in C/Python implementiert
- Api um Python Objekte/Funktionen in C zu implementieren
- Einfach “embeddable”

Performance

schlecht:

```
i = i + 1
```

gut:

```
a = range(20)
a.reverse()
```

hervorragend:

```
import Numeric
a = Array(64000)
a = a + 7
```

Python vs. Lisp - Gemeinsamkeiten

- Dynamisch
- first class Funktionen
- Interaktiv

Python vs. Lisp - Unterschiede

- keine Macros in Python
- Sprache wird konstant weiterentwickelt
- Standardimplementation hochportabel
- Bytecodeinterpreter, aber JIT im Entstehen
- Sehr umfangreiche Standardbibliothek

Conclusion

- Python sehr nahe an Lisp
- Aktive Community
- Für kleine Projekte effizienter, aber auch für grosse Projekte geeignet

Conclusion

- Ein harter Konkurrent

oder

- DIE Einstiegsdroge ?