Info 1 Wiederholung

Daniel Fertmann

24. Januar 2016

Inhaltsverzeichnis

Sequenzen
- Strings
- Tupel
- Listen
- Operationen auf Sequenzen

Identitäten und Objekte

Baum
- Binärbaum/Suchbäume
- Traversierung

Tests/Debuggen
- Unittests
- doctests
- Pytests

Mengen und Dictionaries
- Dictionaries

Sequenzen? (10)

Was sind Sequenzen ???

Strings

```
s = "hello" + " " + "world"
```

Strings

```
s = "hello" + " " + "world"
s = "spam" * 6
```

Strings

```
s = "hello" + " " + "world"
s = "spam" * 6
"hallo8welt".split("8")
```

Strings

```
s = "hello" + " " + "world"
s = "spam" * 6
"hallo8welt".split("8")
s = 'string'
```

Strings

```
s = "hello" + " " + "world"
s = "spam" * 6
"hallo8welt".split("8")
s = 'string'
```

Tupel 1

$t = (2,3,4)$

Tupel 1

t = (2,3,4)

t = 2,3,4

Tupel 1

t = (2,3,4)

t = 2,3,4

t = ("otto",(8,"Zahnweh"))

Tupel 1

t = (2,3,4)

t = 2,3,4

t = ("otto",(8,"Zahnweh"))

t = 4,

Tupel 1

t = (2,3,4)

t = 2,3,4

t = ("otto",(8,"Zahnweh"))

t = 4,

t = (4,)

Tupel 1

```
t = (2,3,4)
t = 2,3,4
t = ("otto",(8,"Zahnweh"))
t = 4,
t = (4,)
t1, t2 = 3, 4
```

Tupel 2

```
def foo():
    a = 2
    b = 3
    return a,b

f,g = foo()
h = foo()
```

Listen

```
t = [2,3,4]
```

Listen

```
t = [2,3,4]
t = ["otto" ,[8, "Zahnweh"]]
```

Listen

t = [2,3,4]

t = ["otto" ,[8, "Zahnweh"]]

t = []

Listen

t = [2,3,4]

t = ["otto" ,[8, "Zahnweh"]]

t = []

t = (4,)

Listen

t = [2,3,4]

t = ["otto" ,[8, "Zahnweh"]]

t = []

t = (4,)

Bsp.

```
>>> [a, (b, c), (d, e), f] = (42, (6, 9), "do", [1, 2, 3])
>>> print(a, "*", b, "*", c, "*", d, "*", e, "*", f)
```

Bsp.

```
>>> [a, (b, c), (d, e), f] = (42, (6, 9), "do", [1, 2, 3])
>>> print(a, "*", b, "*", c, "*", d, "*", e, "*", f)
42 * 6 * 9 * d * o * [1, 2, 3]
```

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:
Verkettung: $"Gambol" + "putty" == "Gambolputty"$

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:

Verkettung: $"Gambol" + "putty" == "Gambolputty"$

Wiederholung: $2 * "spam" == "spamspam"$

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:

Verkettung: $\text{"Gambol"} + \text{"putty"} == \text{"Gambolputty"}$

Wiederholung: $2 * \text{"spam"} == \text{"spamspam"}$

Indizierung: $\text{"Python"}[1] == \text{"y"}$

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:

Verkettung: `"Gambol" + "putty" == "Gambolputty"`

Wiederholung: `2 * "spam" == "spamspam"`

Indizierung: `"Python"[1] == "y"`

Mitgliedschaftstest: `17 in [11,13,17,19]`

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:

Verkettung: `"Gambol" + "putty" == "Gambolputty"`

Wiederholung: `2 * "spam" == "spamspam"`

Indizierung: `"Python"[1] == "y"`

Mitgliedschaftstest: `17 in [11,13,17,19]`

Slicing: `"Monty Python's Flying Circus"[6:12] == "Python"`

Operationen auf Sequenzen

Sequenztypen untersttzen die folgenden Operationen:

Verkettung: `"Gambol" + "putty" == "Gambolputty"`

Wiederholung: `2 * "spam" == "spamspam"`

Indizierung: `"Python"[1] == "y"`

Mitgliedschaftstest: `17 in [11,13,17,19]`

Slicing: `"Monty Python's Flying Circus"[6:12] == "Python"`

Iteration: `for x in "egg"`

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
```

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
>>>["a","b"]+["c","d"]
['a','b','c','d']
```

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
>>>["a","b"]+["c","d"]
['a','b','c','d']
s = "hallo"
s  [0][0][0][0][0][0]
```

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
>>>["a","b"]+["c","d"]
['a','b','c','d']
s = "hallo"
s  [0][0][0][0][0][0]
'h'
```

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
>>>["a","b"]+["c","d"]
['a','b','c','d']
s = "hallo"
s  [0][0][0][0][0][0]
'h'
primes = [2, 3, 6, 7, 11]
>>> primes[2] = 5
>>> print(primes)
```

```
>>> primes = (2, 3, 5, 7, 11, 13)
>>>print(primes[1], primes[-1])
>>>["a","b"]+["c","d"]
['a','b','c','d']
s = "hallo"
s  [0][0][0][0][0][0]
'h'
primes = [2, 3, 6, 7, 11]
>>> primes[2] = 5
>>> print(primes)
[2, 3, 5, 7, 11]
```

```
>>> zahlen = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> zahlen[1:7:2]
```

```
>>> zahlen = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> zahlen[1:7:2]
[1, 3, 5]
>>>zahlen[::-1]
```

```
>>> zahlen = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> zahlen[1:7:2]
[1, 3, 5]
>>>zahlen[::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
zahlen [1:5] = ['spam','spam']
```

Identitäten und Objekte(10)

```
>>> max([1, 23, 42, 5])
42
>>>sum([1, 23, 42, 5])
71
```

Identitäten und Objekte(10)

```
>>> max([1, 23, 42, 5])
42
>>>sum([1, 23, 42, 5])
71
>>> "spam".index("a")
2
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
```

Identitäten und Objekte(10)

```
>>> max([1, 23, 42, 5])
42
>>>sum([1, 23, 42, 5])
71
>>> "spam".index("a")
2
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
(False, False, True)
>>> id(x), id(y), id(z)
```

Identitäten und Objekte(10)

```
>>> max([1, 23, 42, 5])
42
>>>sum([1, 23, 42, 5])
71
>>> "spam".index("a")
2
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
(False, False, True)
>>> id(x), id(y), id(z)
(1076928940, 1076804076, 1076804076)
>>> x == y, x is y
```

Identitäten und Objekte(10)

```
>>> max([1, 23, 42, 5])
42
>>>sum([1, 23, 42, 5])
71
>>> "spam".index("a")
2
>>> x = ["ham", "spam", "jam"]
>>> y = ["ham", "spam", "jam"]
>>> z = y
>>> x is y, x is z, y is z
(False, False, True)
>>> id(x), id(y), id(z)
(1076928940, 1076804076, 1076804076)
>>> x == y, x is y
```

Identitäten und Objekte

```
>>> x = [ 0, 1 , 2]
>>> y = x
>>> x += [ 3 ]
>>> print(x)
```

Identitäten und Objekte

```
>>> x = [ 0, 1 , 2]
>>> y = x
>>> x += [ 3 ]
>>> print(x)
[0,  1,  2,  3 ]
>>> l1 = [1, 3, 5]
>>> l1[2] = l1
>>> print(l1)
```

Identitäten und Objekte

```
>>> x = [ 0, 1 , 2]
>>> y = x
>>> x += [ 3 ]
>>> print(x)
[0,  1,  2,  3 ]
>>> l1 = [1, 3, 5]
>>> l1[2] = l1
>>> print(l1)
[1,  3,  [1,  3,  [ ... ]]]
```

Baum(11)

Was ist ein Baum ???

Baum

Als Baum wird im allgemeinen Sprachgebrauch eine verholzte Pflanze verstanden, die aus einer Wurzel, einem daraus emporsteigenden, hochgewachsenen Stamm und einer belaubten Krone besteht.
Wikipedia

Baum

Wurzel
Knoten
innerer Knoten
Blatt
Binärbaum
Suchbaum
Ausdrucksbaum

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.
Jeder Knoten wird durch eine Liste reprsentiert.

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.
Jeder Knoten wird durch eine Liste reprsentiert.
Die Markierung ist das erste Element der Liste.

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.
Jeder Knoten wird durch eine Liste reprsentiert.
Die Markierung ist das erste Element der Liste.
Der linke Teilbaum ist das zweite Element.

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.
Jeder Knoten wird durch eine Liste reprsentiert.
Die Markierung ist das erste Element der Liste.
Der linke Teilbaum ist das zweite Element.
Der rechte Teilbaum ist das dritte Element.

Binrbume durch Listen reprsentieren

Der leere Baum wird duch None reprsentiert.
Jeder Knoten wird durch eine Liste reprsentiert.
Die Markierung ist das erste Element der Liste.
Der linke Teilbaum ist das zweite Element.
Der rechte Teilbaum ist das dritte Element.

Traversierung

Was ist Traversierung?

Traversierung

Was ist Traversierung?
Welche Methoden gibt es?

Traversierung

3 Vorgehensweisen (Traversierungen) sind blich:
-Pre-Order
(Hauptreihenfolge): Zuerst der Knoten selbst, dann der linke, danach der rechte Teilbaum
-Post-Order
(Nebenreihenfolge): Zuerst der linke, danach der rechte Teilbaum, zum Schluss der Knoten selbst
-In-Order
(symmetrische Reihenfolge): Zuerst der linke Teilbaum, dann der Knoten selbst, danach der rechte Teilbaum
Manchmal betrachtet man auch
-Reverse In-Order (anti-symmetrische Reihenfolge): Rechter Teilbaum,
Knoten, dann linker Teilbaum Auch das Besuchen nach Tiefenlevel von links nach rechts (level-order) ist denkbar

Debuggen(12)

Welche Fehler gibt es?

Debuggen(12)

Welche Fehler gibt es?
Synatx, Laufzeit, Semantik

Debuggen(12)

Welche Fehler gibt es?
Synatx, Laufzeit, Semantik
"str" + C
print (...)
Extremfälle
Debugger?

Tests

Gedanke

Tests

Gedanke
Welche Tests gibt es?

Tests

Gedanke
Welche Tests gibt es?
Unittests
doctetst
Pytests

Unittests

-

doctests

```
import doctest
def expreval(tree):
"""Takes an integer expression tree and evaluates it.
>>> expreval([5, None, None])
5
>>> expreval(['*', [7, None, None], [6, None, None]])
42
"""
...
```

Pytests

```
1 if __name__ == "__main__":
      doctest.testmod()
```

Pytests

```
import pytest
...
def test_expreval_b():
    """Test of expreval that fails."""
    expr = ['*', ['+', [3, None, None],
                 [5, None, None]],[6, None, None]]
    assert expreval(expr) == 42

if __name__ == "__main__":
    # -v switches verbose on
    pytest.main("-v %s" % __file__)
```

Dictionaries

{key1: value1, key2: value2, ... }

dict(key1=value1, key2=value2, ...)

dict([(key1, value1), (key2, value2), ...])

Wird value (und das Komma) weggelassen, wird None verwendet.

Dictionaries

```
{key1: value1, key2: value2, ... }
dict(key1=value1, key2=value2, ... )
dict([(key1, value1), (key2, value2), ... ])
```

Wird value (und das Komma) weggelassen, wird None verwendet.

```
>>> english = ["red", "blue", "green"]
>>> german = ["rot", "blau", "gr"un"]
>>> dict(zip(english, german))
```

Dictionaries

```
{key1: value1, key2: value2, ... }
dict(key1=value1, key2=value2, ... )
dict([(key1, value1), (key2, value2), ... ])
```

Wird value (und das Komma) weggelassen, wird None verwendet.

```
>>> english = ["red", "blue", "green"]
>>> german = ["rot", "blau", "gr"un"]
>>> dict(zip(english, german))
{'red':'rot','green':'grn','blue':'blau'}
>>> dict.fromkeys("abc")
{'a': None,'c': None,'b': None}
>>> dict.fromkeys(range(3), "eine Zahl")
{0:'eine Zahl', 1:'eine Zahl', 2:'eine Zahl'}
```

```
key in d
len(d)
```

```
key in d
len(d)
>>> en_de={'red':'rot','green':'gr"un','blue':'blau'}
>>> en_sw = en_de
>>> en_sw['green'] ='gr"aa'
>>> en_de['green']
```

```
key in d
len(d)
>>> en_de={'red':'rot','green':'gr"un','blue':'blau'}
>>> en_sw = en_de
>>> en_sw['green'] ='gr"aa'
>>> en_de['green']
'gr"aa'
>>> en_de={'red':'rot','green':'gr"un','blue':'blau'}
>>> en_sw = en_de.copy()
>>> en_sw['green'] ='gr"aa'
>>> en_de['green']
```

```
key in d
len(d)
>>> en_de={'red':'rot','green':'gr"un','blue':'blau'}
>>> en_sw = en_de
>>> en_sw['green'] ='gr"aa'
>>> en_de['green']
'gr"aa'
>>> en_de={'red':'rot','green':'gr"un','blue':'blau'}
>>> en_sw = en_de.copy()
>>> en_sw['green'] ='gr"aa'
>>> en_de['green']
'gr"un'
```

ENDE