Vorlesung 06.12.2019

Inhaltsverzeichnis

Kontrollstrukturen

Die Möglichkeit, den Programmablauf zu steuern, ist für imperative Programmierung unabdinglich. Wie bereits von Matlab bekannt, gibt es auch in Python (und damit Sage) mehrere solche Konstrukte, jeweils mit einem *Doppelpunkt* abgeschlossen.

In Python ist der bedingte Block jeweils *einzurücken* - diese Einrückung erhöht die Lesbarkeit und macht ein abschließendes "end" unnötig. Der Block endet, wenn die Einrückung endet.

Kontrollstrukturen können beliebig verschachtelt werden, wobei jeweils auf die entsprechende Ein- sowie Ausrückung geachtet werden muss.

Kontrollstruktur: if-Verzweigung

Die grundlegendste Kontrollstruktur ist die einfache **if**-Verzweigung: Nur wenn die *Bedingung* erfüllt ist, wird der eingerückte Block ausgeführt.

In [2]:
```
if 1 < 0:
    print("1 ist kleiner als 0!")

if 0 < 1:
    print("0 ist kleiner als 1!")
```

Out[2]:
```
0 ist kleiner als 1!
```

Falls mehrere mögliche Fälle überprüft werden sollen, kann mittels **elif** eine weitere Bedingung angegeben werden.
Diese Bedingung wird nur überprüft, wenn die zuvorige **if**-Bedingung sowie etwaige vorherige **elif**-Bedingungen nicht erfüllt waren.

Eine **else**-Bedingung funktioniert wie eine **elif**-Bedingung, die immer zutreffend ist.
Also: sofern die dazugehörigen **if**- und **elif-**Bedingungen nicht erfüllt waren, wird der eingerückte Block ausgeführt.

In [3]:
```
a = 3
b = 1
if a == b:
    print("a ist gleich b!")
elif a <= b:
    print("a ist kleiner als b!")
elif b <= a:
    print("b ist kleiner als a!")
else:
    print("Ist <= hier etwa keine Totalordnung?")
```

Out[3]:
```
b ist kleiner als a!
```

Bedingungen für Fortgeschrittene

Wahrheitswerte ("wahr" oder "falsch") werden in der Informatik für gewöhnlich als *boolean* bezeichnet. In Python heißt die "wahr"-Konstante **True** und die "falsch"-Konstante **False**.
Jede *Bedingung* ist in Wahrheit ein solcher *boolean*-Wert. Falls der angegebene Ausdruck noch kein *boolean*-Wert sein sollte, wird er in einen solchen umgewandelt. Hierbei wird beispielsweise aus der Null der Wert **False** und aus jeder anderen Zahl der Wert **True**.

In [64]:
```
parent(True)
```

Out[64]:
```
<type 'bool'>
```

In [65]:
```
parent(1<2)
```

Out[65]:
```
<type 'bool'>
```

In [66]:
```
parent(42)
```

Out[66]:
```
Integer Ring
```

In [67]:
```
bool(42)
```

Out[67]:
```
True
```

In [68]:
```
bool(0)
```

Out[68]:
```
False
```

Mehrere *boolean*-Werte können miteinander verknüpft werden, um eine komplexere Bedingung zu bilden.
Hierfür gibt es die üblichen Operatoren **not** (unär) sowie **and** und **or** (binär). Um die Abfolge explizit zu machen, können **Klammern** verwendet werden.

In [7]:
```
a = 12
b = 30
c = 35

if not 5.divides(a):
    print("5 ist kein Teiler von a!")

if 6.divides(b) or 6.divides(c):
    print("6 teilt b oder c!")

if 6.divides(b) and (not 6.divides(c)):
    print("6 teilt b, aber nicht c!")
```

Out[7]:
```
5 ist kein Teiler von a!
6 teilt b oder c!
6 teilt b, aber nicht c!
```

Kontrollstruktur: while-Schleife

Wie bereits aus Matlab bekannt, ähnelt die **while**-Schleife in ihrer Natur der **if**-Verzweigung.
Der Unterschied besteht darin, dass nach jedem Durchlauf des eingerückten Blocks wiederum die *Schleifenbedingung* überprüft und gegebenenfalls der Block erneut durchlaufen wird.

In [9]:
```
i = 2
while i < 10000:
    print(i)
    i = i * 2
```

Out[9]:
```
2
4
8
16
32
64
128
256
512
1024
2048
4096
8192
```

Kontrollstruktur: for-Schleife

Die strukturiertere Alternative hierzu bietet die **for**-Schleife, die den eingerückten Block für jeden Wert (in einer Liste oder ähnlichen *iterierbaren* Objekten) einmal durchläuft.
(Im weiteren Verlauf der Lehrveranstaltung werden andere iterierbare Objekte vorgestellt werden.)

In [11]:
```
L = [4, 8, 15, 16, 23, 42]
for n in L:
    print(n)
```

Out[11]:
```
4
8
15
16
23
42
```

Listen für Fortgeschrittene (Teil 1)

Um längere Listen zu erzeugen, gibt es verschiedene Kurzschreibweisen:

Implizite Schrittweite 1

In [69]:
```
range(0,10)
```

Out[69]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [70]:
```
[0,..,10]
```

Out[70]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Explizite Schrittweite

In [76]:
```
range(0,10,2)
```

Out[76]: [0, 2, 4, 6, 8]

In [71]:
```
[0,2,..,10]
```

Out[71]: [0, 2, 4, 6, 8, 10]

Auch möglich - negative Schrittweite!

In [78]:
```
range(10,0,-2)
```

Out[78]: [10, 8, 6, 4, 2]

In [13]:
```
[10,8,..,0]
```

Out[13]: [10, 8, 6, 4, 2, 0]

Listenelemente sind nicht auf Zahlen beschränkt - beliebige Datentypen können in derselben Liste gespeichert werden.

Listen selbst sind auch eine Form von Datentyp - so können beispielsweise Listen in Listen enthalten sein.

In [17]:
```
A = [20, "foo", True, [log(15), "bar"]]
print(A)
for el in A:
    print(parent(el))
```

Out[17]:
```
[20, 'foo', True, [log(15), 'bar']]
Integer Ring
<type 'str'>
<type 'bool'>
<type 'list'>
```

Funktionen

Wie in fast jeder imperativen Programmiersprache kann man auch in Python kleine "Teilprogramme" - so genannte *Funktionen*, nicht zu verwechseln mit dem mathematischen Begriff - definieren.
Dies erlaubt es, den Programmcode gut lesbar und übersichtlich zu halten, indem idente Operationen nicht mehrmals ausprogrammiert werden müssen.

In Python werden Funktionen mit dem Schlüsselwort **def** definiert.

In [19]:
```
def print_list_types(l):
    print("======")
    for e in l:
        print(parent(e), e)

print_list_types([1,2,"foo",1/2])
print_list_types([["abra","cadabra"], log(e), 15^51])
```

Out[19]:
```
======
(Integer Ring, 1)
(Integer Ring, 2)
(<type 'str'>, 'foo')
(Rational Field, 1/2)
======
(<type 'list'>, ['abra', 'cadabra'])
(Symbolic Ring, 1)
(Integer Ring, 95643225032107438035511171037228450586553663
0153656005859375)
```

Mittels des Schlüsselworts **return** kann die Funktion frühzeitig beendet werden. Optional kann dem **return** ein Wert nachfolgen - dieser ist dann der *Rückgabewert* der Funktion.

In [21]:
```
def absolute(x):
    if x > 0:
        return x
    else:
        return -x

print(absolute(-42), absolute(15))
```

Out[21]:
```
(42, 15)
```

In [83]:
```
absolute(x)
```

Out[83]:
```
-x
```

Für "unübliche" Funktionsparameter kann ein *Standardwert* vergeben werden. Der Parameter wird dadurch *optional*, d.h. muss nicht mehr angegeben werden.

In [113]:
```
def inc(a,b=1):
    return a+b
```

Out[113]:

In [114]:
```
inc(1)
```

Out[114]:
```
2
```

In [115]:
```
inc(1,3)
```

Out[115]:
```
4
```

Rekursion

Es ist auch erlaubt, innerhalb einer Funktion dieselbe Funktion nochmals aufzurufen. Dies bezeichnet man als *Rekursion*.

In [84]:
```
def ggT(a,b):
    if a < 0:
        return ggT(-a,b)
    if b < 0:
        return ggT(a,-b)
    if a < b:
        return ggT(b,a)
    if b == 0:
        return a
    return ggT(b,a-b)
```

Out[84]:

In [89]:
```
ggT(135,165)
```

Out[89]: 15

Die Fibonaccifolge

In [106]:
```
def fib1(n):
    if n < 2:
        return 1
    else:
        return fib1(n-1)+fib1(n-2)
```

Out[106]:

Das wird sehr langsam.

In [107]:
```
fib1(33)
```

Out[107]: 5702887

Wir zählen mit, wie oft die Funktion aufgerufen wird. Wenn innerhalb einer Funktion eine Variable definiert wird, hat sie nichts mit Variablen außerhalb der Funktion zu tun. Wir bezeichnen solche Variablen als *lokal*.
Falls eine Funktion explizit eine *globale* Variable verändern will, kann diese Variable mittels des Schlüsselworts **global** sichtbar gemacht werden.

In [110]:
```
nsa = []

def fib2(n):
    global nsa
    nsa.append(n)

    if n < 2:
        return 1
    else:
        return fib2(n-1)+fib2(n-2)
```

Out[110]:

In [111]:
```
fib2(10)
len(nsa)
```

Out[111]: 354

Für rekursiv definierte mathematische Funktionen mit hoher Rekursionstiefe (oder einer hohen Anzahl an redundanten Aufrufen mit denselben Parametern während der Berechnung) kann eine Definition als klassische rekursive Funktion oft äußerst langsam werden.
Aus diesem Grund gibt es die Möglichkeit, die Funktion für jeden möglichen Parameterwert nur einmal auszuführen und die Rückgabewerte zu speichern. Dies erreicht man durch **@cached_function**, einen sogenannten *Dekorator*.

In [96]:
```
nsa3 = []
@cached_function
def fib3(n):
    global nsa3
    nsa3.append(n)

    if n < 2:
        return 1
    else:
        return fib3(n-1) + fib3(n-2)
```

Out[96]:

In [31]:
```
fib3(10)
```

Out[31]: 89

In [97]:
```
nsa3
```

Out[97]: [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

Weiters zu beachten ist die von Python vorgegebene *maximale Rekursionstiefe*.

In [117]:
```
fib3(500)
```

Out[117]: WARNING: Output truncated!

```
Exception RuntimeError: 'maximum recursion depth exceeded w
hile calling a Python object' in <type 'exceptions.KeyError
'> ignored
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "_sage_input_19.py", line 10, in <module>
    exec compile(u'open("___code___.py","w").write("# -*- c
oding: utf-8 -*-\\n" + _support_.preparse_worksheet_cell(ba
se64.b64decode("ZmliMyg1MDAp"),globals())+"\\n"); execfile
(os.path.abspath("___code___.py"))' + '\n', '', 'single')
  File "", line 1, in <module>

  File "/tmp/tmp6KSDk_/___code___.py", line 3, in <module>
    exec compile(u'fib3(_sage_const_500 )' + '\n', '', 'sin
gle')
  File "", line 1, in <module>

  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
```

```
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )

...

    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
```

```
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
```

```
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
  File "/tmp/tmp2nYCDi/___code___.py", line 12, in fib3
    return fib3(n-_sage_const_1 ) + fib3(n-_sage_const_2 )
  File "sage/misc/cachefunc.pyx", line 1005, in sage.misc.c
achefunc.CachedFunction.__call__ (build/cythonized/sage/mis
c/cachefunc.c:6300)
```

Sie kann schrittweise umgangen werden:

In [120]:
```
fib3(250)
```

Out[120]:
```
12776523572924732586037033894655031898659556447352249
```

In [121]:
```
fib3(500)
```

Out[121]:
```
2255915161619363308725126950360720720460113249137581905886388
8664184746277386868834050159870527969684986 26
```

oder bei Bedarf mittels *sys.setrecursionlimit(depth)* erhöht werden.

In [122]:
```
sys.setrecursionlimit(500)
```

Out[122]:

Auch die bekannten **Operatoren**, also z.B. "+" und "-", sind nichts anderes als Funktionen, die zwei Parameter (<-> zwei Seiten des Operators) erwarten.

Falls man diese Funktionen direkt aufrufen möchte, findet man sie unter *operator*, also z.B. *operator.add* und *operator.sub*.

In [44]:
```
operator.add(operator.sub(5,2),operator.sub(9,4))     # die
ser Ausdruck ist äquivalent zu (5-2) + (9-4)
```

Out[44]: 8