lektion3

November 5, 2025

Inhalt

1 Lektion 3

1.1 Wiederholung

Die Befehle

```
from sympy import *
init_printing()
```

sorgen dafür, dass alle Sympyfunktionen geladen werden und dass die Ausgabe hübsch ist.

[1]:
```
from sympy import *
init_printing()
```

[2]:
```
x, n = symbols('x n')
```

[3]:
```
cosc = (cos(x) - 1) / x
cosc
```

[3]:
$$\frac{\cos(x) - 1}{x}$$

[4]:
```
cosc.subs(x, 0)  # Achtung Falle
```

[4]:
NaN

[5]:
```
cosc.evalf(subs={x: 0})
```

[5]:
$\tilde{\infty}$

1.2 Grenzwerte

Wir schauen uns Grenzwerte von Folgen oder Funktionen an.

$$\lim_{x \to a} f(x)$$

[6]:
```
cosc.limit(x, 0)
```

[6]:
0

[7]:
```
limit(cosc, x, 0)
```

[7]:
0

[8]:
```
Limit(cosc, x, 0)  # träger Operator
```

[8]:
$$\lim_{x \to 0^+} \left(\frac{\cos(x) - 1}{x} \right)$$

[9]:
```
L = Limit(1 / x, x, 0)
L
```

[9]:
$$\lim_{x \to 0^+} \frac{1}{x}$$

[10]:
```
L.doit()
```

[10]:
∞

[11]:
```
Limit(1 / x, x, 0, '-').doit()
```

[11]:
$-\infty$

[12]:
```
b = factorial(n) / sqrt(n) * (E / n)**n
b
```

[12]:
$$\frac{\left(\frac{e}{n}\right)^{n} n!}{\sqrt{n}}$$

[13]:
```
L = Limit(b, n, oo)
L
```

[13]:
$$\lim_{n \to \infty} \left(\frac{\left(\frac{e}{n}\right)^{n} n!}{\sqrt{n}} \right)$$

[14]:
```
L.doit()
```

[14]:
$$\sqrt{2}\sqrt{\pi}$$

[15]:
```
bb = b.evalf(subs={n: 10000})
bb
```

[15]:
$$2.50664916328698$$

[16]:
```
N(bb - sqrt(2*pi))
```

[16]:
$$2.08886559842248 \cdot 10^{-5}$$

1.3 Summen und Reihen

$$\sum_{j=m}^{n} a_n$$

n darf hierbei auch ∞ sein.

[17]:
```
m = S('m')
s = Sum(m, (m, 0, n))
s
```

[17]:
$$\sum_{m=0}^{n} m$$

[18]:
```
s.doit()
```

[18]:
$$\frac{n^2}{2} + \frac{n}{2}$$

[19]:
```
s.doit().factor()
```

[19]:
$$\frac{n\,(n+1)}{2}$$

[20]:
```
g = Sum(x**m, (m, 0, n))
g
```

[20]:
$$\sum_{m=0}^{n} x^{m}$$

[21]:
```
g.doit()
```

[21]:
$$\begin{cases} n + 1 & \text{for } x = 1 \\ \frac{1 - x^{n+1}}{1 - x} & \text{otherwise} \end{cases}$$

[22]:
```
g = Sum(x**m, (m, 0, oo)) # unendlich oo "zwei kleine o"
g
```

[22]:
$$\sum_{m=0}^{\infty} x^{m}$$

[23]:
```
g.doit()
```

[23]:
$$\begin{cases} \frac{1}{1 - x} & \text{for } |x| < 1 \\ \sum_{m=0}^{\infty} x^{m} & \text{otherwise} \end{cases}$$

[24]:
```
g.doit().args
```

[24]:
$$\left(\left(\frac{1}{1 - x}, \ |x| < 1 \right), \ \left(\sum_{m=0}^{\infty} x^{m}, \ \text{True} \right) \right)$$

[25]:
```
g.doit().args[0]
```

[25]:
$$\left(\frac{1}{1 - x}, \ |x| < 1 \right)$$

[26]:
```
g.doit().args[0][0]
```

[26]:
$$\frac{1}{1 - x}$$

[27]:
```
a = Sum((-1)**(m + 1) / m, (m, 1, oo))
a
```

[27]:
$$\sum_{m=1}^{\infty} \frac{(-1)^{m+1}}{m}$$

[28]:
```
a.doit()
```

[28]:
$$\log\left(2\right)$$

[29]:
```
s = Sum(1 / m**2, (m, 1, oo))
s
```

[29]:
$$\sum_{m=1}^{\infty} \frac{1}{m^2}$$

[30]:
```
s.doit()
```

[30]:
$$\frac{\pi^2}{6}$$

1.4 Ableitungen

[31]:
```
f = x**n
```

[32]:
```
f.diff(x)
```

[32]:
$$\frac{nx^n}{x}$$

[33]:
```
f.diff(x).powsimp()
```

[33]:
$$nx^{n-1}$$

[34]:
```
f.diff(x,x,x)
```

[34]:
$$\frac{nx^n \left(n^2 - 3n + 2\right)}{x^3}$$

[35]:
```
f.diff(x, x, x).powsimp().factor()
```

[35]:
$$nx^{n-3} \left(n - 2\right) \left(n - 1\right)$$

[36]:
```
f.diff(x, 3).powsimp().factor()
```

[36]:
$$nx^{n-3} \left(n - 2\right) \left(n - 1\right)$$

[37]:
```
f.diff(x, 0)
```

[37]:
$$x^n$$

[38]:
```
f.diff()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[38], line 1
----> 1 f.diff()

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/core/expr.py:3606, in Expr.diff(self, *symbols, **assumptions)
```

```
   3604 def diff(self, *symbols, **assumptions):
   3605     assumptions.setdefault("evaluate", True)
-> 3606     return _derivative_dispatch(self, *symbols, **assumptions)

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/core/function.py:1938, in _derivative_dispatch(expr, *variables,␣
 ↪**kwargs)
   1936     from sympy.tensor.array.array_derivatives import ArrayDerivative
   1937     return ArrayDerivative(expr, *variables, **kwargs)
-> 1938 return Derivative(expr, *variables, **kwargs)

File /local/home/schaedle/miniconda3/envs/compla24/lib/python3.12/site-packages
 ↪sympy/core/function.py:1284, in Derivative.__new__(cls, expr, *variables,␣
 ↪**kwargs)
   1279             raise ValueError(filldedent('''
   1280                 Since there are no variables in the expression,
   1281                 the variable(s) of differentiation must be supplied
   1282                 to differentiate %s''' % expr))
   1283         else:
-> 1284             raise ValueError(filldedent('''
   1285                 Since there is more than one variable in the
   1286                 expression, the variable(s) of differentiation
   1287                 must be supplied to differentiate %s''' % expr))
   1289 # Split the list of variables into a list of the variables we are diff
   1290 # wrt, where each element of the list has the form (s, count) where
   1291 # s is the entity to diff wrt and count is the order of the
   1292 # derivative.
   1293 variable_count = []

ValueError:
Since there is more than one variable in the expression, the
variable(s) of differentiation must be supplied to differentiate x**n
```

[39]:
```
g = exp(x)  # das geht, da in dem Ausdruck exp(x) nur ein Symbol vorkommt
g.diff()
```

[39]: e^{x}

[40]:
```
y = symbols('y')
```

[41]:
```
g = exp(x*y)
```

[42]:
```
g.diff(x, x, y, y, y)
```

[42]: $x\left(x^{2}y^{2}+6xy+6\right)e^{xy}$

[43]:
```
g.diff(x, 2, y, 3)
```

[43]:

$$x\left(x^2y^2+6xy+6\right)e^{xy}$$

1.5 Grenzwert mit der Regel von l'Hospital

[44]:
```
a = (cos(pi * x) - 1 + pi * x / 2) / x
```

[45]:
```
na = numer(a)  # numer gibt es nicht als methode
da = denom(a)
```

[46]:
```
na.diff(x)
```

[46]:
$$-\pi\sin\left(\pi x\right)+\frac{\pi}{2}$$

[47]:
```
n1 = na.diff(x).subs(x, 0)  # hier kann man gefahrlos substituieren
n1
```

[47]:
$$\frac{\pi}{2}$$

[48]:
```
d1 = da.diff(x).subs(x, 0)
d1
```

[48]:
$$1$$

[49]:
```
l = n1 / d1
l
```

[49]:
$$\frac{\pi}{2}$$

[50]:
```
l == a.limit(x, 0)
```

[50]:
```
True
```

1.6 Unbestimmte Integrale

[51]:
```
f = x**n
f
```

[51]:
$$x^n$$

[52]:
```
I = Integral(f, x)  # Achtung jetzt haben wird die imaginäre Einheite␣
↪überschrieben
I
```

[52]:
$$\int x^n\,dx$$

[53]:
```
F = I.doit()
F
```

[53]:
$$\begin{cases} \frac{x^{n+1}}{n+1} & \text{for } n \neq -1 \\ \log(x) & \text{otherwise} \end{cases}$$

[54]:
```
F.diff(x).simplify()
```

[54]:
$$\begin{cases} x^{n} & \text{for } n > -1 \vee n < -1 \\ \frac{1}{x} & \text{otherwise} \end{cases}$$

[55]:
```
g = 1 / (1 + x**4)
I2 = Integral(g, x)
I2
```

[55]:
$$\int \frac{1}{x^4 + 1}\, dx$$

[56]:
```
G = I2.doit()
G
```

[56]:
$$-\frac{\sqrt{2}\log\left(x^2 - \sqrt{2}x + 1\right)}{8} + \frac{\sqrt{2}\log\left(x^2 + \sqrt{2}x + 1\right)}{8} + \frac{\sqrt{2}\operatorname{atan}\left(\sqrt{2}x - 1\right)}{4} + \frac{\sqrt{2}\operatorname{atan}\left(\sqrt{2}x + 1\right)}{4}$$

[57]:
```
G.diff(x) == g
```

[57]:
```
False
```

[58]:
```
G.diff(x)
```

[58]:
$$-\frac{\sqrt{2}\left(2x - \sqrt{2}\right)}{8\left(x^2 - \sqrt{2}x + 1\right)} + \frac{\sqrt{2}\left(2x + \sqrt{2}\right)}{8\left(x^2 + \sqrt{2}x + 1\right)} + \frac{1}{2\left(\left(\sqrt{2}x + 1\right)^2 + 1\right)} + \frac{1}{2\left(\left(\sqrt{2}x - 1\right)^2 + 1\right)}$$

[59]:
```
G.diff(x).ratsimp()
```

[59]:
$$\frac{1}{x^4 + 1}$$

Um die Gleichheit zweier Ausdrücke zur testen vereinfacht man besser die Differenz, statt mit == zu testen.

[60]:
```
simplify(G.diff(x) - g)
```

[60]:
$$0$$

1.7 Bestimmte Integrale

[61]:
```
Integral(x * (x - n), (x, 0, n))
```

[61]:
$$\int_0^n x\,(-n + x)\, dx$$

[62]:
```
Integral(x * (x - n), (x, 0, n)).doit()
```

[62]:
$$-\frac{n^3}{6}$$

[63]:
```
I3 = Integral(g, (x, -oo, oo))
I3
```

[63]:
$$\int_{-\infty}^{\infty} \frac{1}{x^4+1}\,dx$$

[64]:
```
I3.doit()
```

[64]:
$$\frac{\sqrt{2}\pi}{2}$$

[65]:
```
G.limit(x, oo) - G.limit(x, -oo)
```

[65]:
$$\frac{\sqrt{2}\pi}{2}$$

1.8 Wenn sympy nicht mehr weiterweiß

[66]:
```
Integral(x/(exp(x)-1), (x, 1, 5)).doit()
```

[66]:
$$\int_{1}^{5} \frac{x}{e^x-1}\,dx$$

[67]:
```
Integral(x/(exp(x)-1),(x,1,5)).evalf()
```

[67]:
0.826876354388482

[68]:
```
Integral(x**(-x),(x,0,1)).doit()
```

[68]:
$$\int_{0}^{1} x^{-x}\,dx$$

[69]:
```
N(Integral(x**(-x),(x,0,1)))
```

[69]:
1.29128599706266

1.9 For Schleifen

[70]:
```
list(range(10))
```

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

[71]:
```
k = 0
for j in range(10):
    print('innen\t', j)
    k =  k + j
print('aussen\t', j + 100, k)
```

```
innen    0
innen    1
innen    2
innen    3
innen    4
innen    5
innen    6
innen    7
innen    8
innen    9
aussen   109 45
```

[72]:
```
from sympy import Rational
```

Blöcke werden durch Einrückung/Ausrückung festgelegt

[73]:
```
for p in range(-3, 3, 2):
    for q in range(1, 4):
        print(f"p = {p}, q = {q}")
        display(Rational(p, q))
```

```
p = -3, q = 1
```

-3

```
p = -3, q = 2
```

$-\frac{3}{2}$

```
p = -3, q = 3
```

-1

```
p = -1, q = 1
```

-1

```
p = -1, q = 2
```

$-\frac{1}{2}$

```
p = -1, q = 3
```

$-\frac{1}{3}$

```
p = 1, q = 1
```

$$1$$

```
p = 1, q = 2
```

$$\frac{1}{2}$$

```
p = 1, q = 3
```

$$\frac{1}{3}$$

[74]:
```
summe = 0
N = 20
for n in range(1, N):
    summe += Rational(1, n)
summe
```

[74]:
$$\frac{275295799}{77597520}$$

1.10 Einschub

Mit f-String erzeugt man in Python einen formatierten String. Der Python Code in { } Klammern wird dabei ausgewertet.

[75]:
```
n = 10
f'abc {n+1}'
```

[75]:
```
'abc 11'
```

einfache Beispiele von https://docs.python.org/3/reference/lexical_analysis.html#f-strings

[76]:
```
a = 3
f"abc{a # This is a comment
+ 3}"
```

[76]:
```
'abc6'
```

[77]:
```
name = "Fred"
f"He said his name is {name!r}."
```

[77]:
```
"He said his name is 'Fred'."
```

[78]:
```
f"He said his name is {repr(name)}."  # repr() is equivalent to !r
```

[78]:
```
"He said his name is 'Fred'."
```

[79]:
```
value = 12.3456789
```

```
f'result: {value:.3e}  {value:.3f}' # formating 'e' scientific notation, 'f' fixed-point notation
```

[79]:
```
'result: 1.235e+01  12.346'
```

[80]:
```
width = 10
precision = 4
value = 12.34567
f"result:{value:{width}.{precision}}"  # nested fields
```

[80]:
```
'result:      12.35'
```

1.11 Vergleiche

[81]:
```
1 == 0  # gleich
```

[81]:
```
False
```

[82]:
```
1 != 0 # ungleich
```

[82]:
```
True
```

[83]:
```
1 <= 2 # kleinergleich
```

[83]:
```
True
```

Weiter gibt es < (kleiner) und > (größer) und >= (größergleich)

1.12 Logische Verknüpfungen

[84]:
```
True, False #boolsche Variablen
```

[84]:
```
(True, False)
```

[85]:
```
not True # Negation
```

[85]:
```
False
```

zweistellige logische Verknüpfungen sind **and** für “und” und **or** für “oder”

[86]:
```
for a in [True, False]:
    for b in [True, False]:
        print(f"{a} or {b} == {a or b}")
```

```
True or True == True
True or False == True
False or True == True
False or False == False
```

alternativ kann man das bitwise & für “und” und | für “oder” verwenden

[87]:
```
print("\n")  # \n 'newline' macht einen Zeilenumbruch in der Ausgabe
for a in [True, False]:
    for b in [True, False]:
        print(f"{a} | {b} == {a|b}")
```

```
True | True == True
True | False == True
False | True == True
False | False == False
```

[88]:
```
1 < 2 < 1  # äquivalent zu  (1 < 2) und (2 < 1)
```

[88]:
```
False
```

[89]:
```
# Vorlesung 30.10
```

1.13 Bedingte Anweisung und Verzweigungen

Bedingte Anweisungen (if Anweisungen) dienen dazu, den Programmfluss unter bestimmten Bedingungen verzweigen zu lassen. Damit wird es möglich während das Programm ausgeführt wird zu entscheiden, ob bestimmte Programmteile ausgeführt werden sollen oder nicht.

```
if bedinung:
    anweisung
```

`bedingung` ist hier ein Ausdruck, der wahr oder falsch ist. Optional kann man weitere Bedingungen angeben, die überprüft werden, falls die erste nicht erfüllt ist.

[90]:
```
a = 42
if 1 < a < 100:
    print(f'richtig: {a}')
```

```
richtig: 42
```

[91]:
```
a = 200
if 1 < a < 100:
    print(f'richtig: {a}')
else:
    print('zu groß oder zu klein ')
```

```
zu groß oder zu klein
```

[92]:
```
a = 0
if 1 < a < 100:
    print(f'richtig: {a}')
```

```
elif a >= 100:
    print('zu groß')
else:
    print('zu klein')
```

```
zu klein
```

1.13.1 Beispiel

Finde das kleinste m so, dass

$$\sum_{j=1}^{m} \frac{1}{j} \geq 5$$

[93]:
```
summe = 0
for m in range(1, 1000000):
    summe += Rational(1, m)
    if summe >= 5:
        break

summe.evalf(), m
```

[93]:
(5.00206827268017, 83)

break bricht eine Schleife ab (bei verschachtelten Schleifen die innerste)

1.14 While Schleife

Der Aufbau einer while Schleife ist wie folgt. Der Block `anweisung1` wird ausgeführt solange `bedingung` True ist.

```
while bedingung:
    anweisung1
```

Wieder kann man mit `break` die Schleife abbrechen oder mit `continue` zum Anfang der Schleife springen.

[94]:
```
summe = 0
m = 0
while summe < 5:
    m += 1
    summe += Rational(1, m)

summe.evalf(), m
```

[94]:
(5.00206827268017, 83)

[95]:
```
a = 1
print(a)
```

```python
while True:
    if a > 10:
        print(f'Fertig {a} ist groß genug')
        break
    a = a + (a + 1)
    print(a)
```

```
1
3
7
15
Fertig 15 ist groß genug
```

[ ]: