aufgabe3

January 18, 2022

0.1 Aufgabe 3

1. Binden Sie das Modul math ein und testen Sie durch die Ausgabe der Variable math.pi ob dies funktioniert hat.

[1]:
```
import math  # Importieren des Moduls "math"
```

0.1.1 Einschub: Aufgabe 4

Definieren Sie eine Funktion my_sin(x) die auf diese Weise $sin(x)$ berechnet und zurück gibt. Setzen Sie dabei in der Funktion n_max fest auf den Wert 12. Testen Sie die Funktion in dem Sie my_sin(x) für verschiedene Werte berechnen.

Eigene Funktionen werden in Python üblicherweise gleich am Anfang des Skripts nach den import-Anweisungen definiert. Verschieben Sie die Zelle mit der Funktionsdefinition entsprechend an den Anfang des Notebooks.

[2]:
```
def my_sin(x):
    n_max = 12
    y = 0
    # range() kann auch rückwärts zählen, die Schrittweite ist dann negativ
    # Beachten Sie, dass der Bereich *exclusiv* der zweiten Grenze angegeben
    # wird, es muss also hier $-1$ als Grenze angegeben werden
    for n in range(n_max, -1, -1):
        y += (-1)**n * x**(2*n + 1) / math.factorial(2*n + 1)
    return(y)
```

[3]:
```
print(my_sin(0))
print(my_sin(math.pi/2))
print(my_sin(math.pi))
print(my_sin(3 * math.pi/2))
```

```
0.0
1.0
2.6645352591003757e-15
-0.9999999998654854
```

0.1.2 Fortsetzung Aufgabe 3

[4]:
```
print(math.pi)  # Testen ob pi nun existiert
```

```
3.141592653589793
```

2. Lassen Sie eine Zahl “Summationsgrenze:” eingeben und weisen Sie die Eingabe der Variable n_max zu. Wandeln Sie den Wert dabei gleich in den Datentyp int um und geben Sie danach den Inhalt der Variable aus. Was passiert wenn Sie als Zahl den Text “Hallo” eingeben?

- Antwort: Geben Sie Text ein, wird Python einen “ValueError” ausgeben, ist die Eingabe keine ganze Zahl kann int() den String nicht umwandeln

[5]:
```
n_max = int(input("Summationsgrenze"))
print(n_max)
```

```
Summationsgrenze 12
```

```
12
```

3. Jetzt kommt eine erste bedingte Anweisung: Testen Sie ob n max $>$ 12 gilt, wenn ja geben Sie eine Fehlermeldung aus.

[6]:
```
if n_max > 12:
    print("Summationsgrenze zu groß gewählt (n_max > 12)")
```

Bei der Konstruktion der Schleife hilft range(). Mit list() kann man explizit in eine Liste umwandeln.

[7]:
```
list(range(10))
# Achtung 1: ohne untere Grenze zählt range() ab 0
# Achtung 2: die obere Grenze ist nicht mehr enthalten, hier [0,10[
# So bekommt man z.B. mit range(10) auch 10 Elemente aber mit den Werten 0-9
```

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

[8]:
```
x = math.pi / 2
y = 0
for n in range(n_max+1):
    y += (-1)**n * x**(2*n + 1) / math.factorial(2*n + 1)
    print(n, ":", y)

print("x = ", x, " -> sin(x) =", y)
```

```
0 : 1.5707963267948966
1 : 0.9248322292886504
2 : 1.0045248555348174
3 : 0.9998431013994987
4 : 1.0000035425842861
5 : 0.999999943741051
6 : 1.0000000006627803
7 : 0.9999999999939768
```

```
8 : 1.0000000000000437
9 : 1.0
10 : 1.0000000000000002
11 : 1.0000000000000002
12 : 1.0000000000000002
x =  1.5707963267948966  -> sin(x) = 1.0000000000000002
```

Für $n > 10$ ändert sich das Ergebnis nicht mehr! Addiere ich auf eine Zahl der Größenordnung 1 eine sehr kleine Zahl, so geht dies durch Rundung verloren:

[9]:
```
1 + 1E-18
```

[9]:
```
1.0
```

0.1.3 Es ist daher günstiger erst die kleinen Summanten zu summieren

`range(start,stop,step)` kann auch rückwärts zählen. Man muss dafür eine negative Schrittweite (`step`) angeben. Beachten Sie wieder, dass die zweite Grenze (`stop`) nicht mehr Teil der Sequenz ist.

[10]:
```
x = math.pi / 2
y = 0
for n in range(n_max, -1, -1):
    y += (-1)**n * x**(2*n + 1) / math.factorial(2*n + 1)
    print(n, ":", y)

print("x = ", x, " -> sin(x) =", y)
```

```
12 : 5.156455176580277e-21
11 : -1.2487430853588762e-18
10 : 2.5589354620068836e-16
9 : -4.3514761126936703e-14
8 : 6.0234209699792556e-12
7 : -6.627800900111671e-10
6 : 5.625894912966807e-08
5 : -3.542584286082416e-06
4 : 0.00015689860050127733
3 : -0.0045248555348174095
2 : 0.07516777071134963
1 : -0.5707963267948966
0 : 1.0
x =  1.5707963267948966  -> sin(x) = 1.0
```

[ ]:
```

```