jupyter

December 10, 2024

Gruppe:
Luca Kässner
Marlene Conrad
Yassin Starzetz

1 Kapitel 1 Teil 1

Übung 1: Schreiben Sie die Zahl 2,99x10^8 m/s in beiden Schreibweisen.

[13]:
```
x = 2.99*10**8  # normal
y = 2.99e8      # kompakt

print(x)
print(y)
```

```
299000000.0
299000000.0
```

Übung 2:

Berechnen Sie die Gravitationskraft zwischen Erde und Sonne, gegeben ist die Masse der Erde mit $5.97 \times 10^{24} kg$ und die der Sonne mit $1.99×10^{30}$ kg . Der Durchschnittsabstand zwischen den beiden Objekten ist gegeben durch $1.5 \times 10^{11} m$ und die Gravitationskonstante beträgt $6.674 \times 10^{-11} N * m^2 \cdot kg^{-2}$.

[14]:
```
G=6.67e-11      # Gravitationskonstante in N*m^2/kg^2
M_E=5.97e24     # Masse Erde in kg
M_S=1.99e30     # Masse Sonne in kg
r=1.5e11        # in meter
F=G*M_E*M_S/r**2
print("Gravitative Anziehungskraft in Newton =",F)
```

```
Gravitative Anziehungskraft in Newton = 3.5218489333333335e+22
```

Übung 3:

Berechnen Sie den Flächeninhalt eines Kreises mit dem Radius=3.0cm in m^2. **Hinweis:** π kann mit `np.pi` aus dem numpy-Modul geladen werden.

[15]:
```
import numpy as np

pi = np.pi
radius = 0.03

area_of_a_circle = pi*radius**2     # Berechnung der Fläche eines Kreises
print(f"{area_of_a_circle} m²")
```

```
0.0028274333882308137 m²
```

Übung 4:

Versuchen Sie die Dokumentation der cos-Funktion im np-Modul mithilfe der verschiedenen Methoden aufzurufen.

[6]:
```
help(np.cos)
```

```
Help on ufunc in module numpy:

cos = <ufunc 'cos'>
    cos(x, /, out=None, *, where=True, casting='same_kind', order='K',
dtype=None, subok=True[, signature])

    Cosine element-wise.

    Parameters
    ----------
    x : array_like
        Input array in radians.
    out : ndarray, None, or tuple of ndarray and None, optional
        A location into which the result is stored. If provided, it must have
        a shape that the inputs broadcast to. If not provided or None,
        a freshly-allocated array is returned. A tuple (possible only as a
        keyword argument) must have length equal to the number of outputs.
    where : array_like, optional
        This condition is broadcast over the input. At locations where the
        condition is True, the `out` array will be set to the ufunc result.
        Elsewhere, the `out` array will retain its original value.
        Note that if an uninitialized `out` array is created via the default
        ``out=None``, locations within it where the condition is False will
        remain uninitialized.
    **kwargs
        For other keyword-only arguments, see the
        :ref:`ufunc docs <ufuncs.kwargs>`.

    Returns
    -------
    y : ndarray
        The corresponding cosine values.
```

```
    This is a scalar if `x` is a scalar.

Notes
-----
If `out` is provided, the function writes the result into it,
and returns a reference to `out`.  (See Examples)

References
----------
M. Abramowitz and I. A. Stegun, Handbook of Mathematical Functions.
New York, NY: Dover, 1972.

Examples
--------
>>> import numpy as np
>>> np.cos(np.array([0, np.pi/2, np.pi]))
array([  1.00000000e+00,   6.12303177e-17,  -1.00000000e+00])
>>>
>>> # Example of providing the optional output parameter
>>> out1 = np.array([0], dtype='d')
>>> out2 = np.cos([0.1], out1)
>>> out2 is out1
True
>>>
>>> # Example of ValueError due to provision of shape mis-matched `out`
>>> np.cos(np.zeros((3,3)),np.zeros((2,2)))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (3,3) (2,2)
```

Übung 5:

In einem rechtwinklichen Dreieck sei y die Gegenkathete des Winkels θ mit y=3.2cm und x=2.4cm die Ankathete. Berechnen Sie den Winkel θ und geben Sie diesen in Radianten sowie Grad an. Nutzen Sie dafür das **math**-Modul.

[7]:
```python
import math
import numpy as np

pi = np.pi
y = 3.2
x = 2.4
radian = math.atan(y/x)      # Winkel zwischen x und y in Radianten
degree = radian*(180/pi)     # Winkel umgerechnet in Grad

print(f"  in Radianten:            {radian}")
print(f"  in Grad:                 {degree}°")
```

```
# gerundet
print("  in Radianten (gerundet): {:.2f}".format(radian))
print("  in Grad (gerundet):      {:.2f}°".format(degree))
```

```
 in Radianten:            0.9272952180016123
 in Grad:                 53.13010235415599°
 in Radianten (gerundet): 0.93
 in Grad (gerundet):      53.13°
```

2 Kapitel 1 Teil 2

Übung 1: Weisen Sie der Variablen x den Wert von π zu (verwenden Sie 3 signifikante Stellen) und finden Sie heraus, zu welchem Datentyp x gehört.

[46]:
```
import numpy as np

x = np.pi
x = round(x, 2) # rundet auf die zweite Nachkommastelle

print(f"Pi:  {x}")
print(f"Typ: {type(x)}")
```

```
Pi:  3.14
Typ: <class 'float'>
```

Übung 2: Mithilfe der Syntax `x[i]` kann das i-te element der Liste x aufgerufen werden. Finden Sie heraus, wie man auf die erste der beiden oben erstellten Listen zugreift. Versuchen Sie Elemente mit negativen Indizes aufzurufen, z.B. `x[-1]`. Beschreiben Sie mit eigenen Worten was passiert ist.

[27]:
```
list_1= [1, 2, 3, 4]
list_2 = [1, 2, "car", "boat"]

print(list_1[0])
print(list_2[0])
print(list_1[-1], list_1[-2])
```

```
1
1
4 3
```

Beim Index [-1] wird das letzte Element der Liste zurückgegeben. Ein negativer Index, indexiert eine Liste sozusagen von hinten nach vorne.

Übung 3:

Versuchen Sie den **print**-Befehl außerhalb des **for**-Blocks zu schreiben.

[62]:
```
s=0.0
for k in range(1,101):
```

```
    s+=1/k
print("Die Summe ist:" , s) # gibt das Ergebnis der ganzen Schleife zurück
```

```
Die Summe ist: 5.187377517639621
```

Es wird nur noch das Resultat des For-Loops ausgegeben.

Übung 4:

Definieren Sie eine Funktion, um den Durchschnitt von vier Zahlen zu bestimmen.

[34]:
```
def mean(numbers: list) -> float:
    n = len(numbers)        # Anzahl der Zahlen
    sum = 0
    for number in numbers:  # Schleife welche die Summe der Zahlen berechnet
        sum += number
    med = sum/n             # Durchschnitt ist Summe / Anzahl
    return med

# Andere Schreibweise mit built-in Funktionen
def mean2(numbers: list) -> float:
    return sum(numbers) / len(numbers)

# Wenn nur Listen mit der Länge 4 erlaubt sind, kann man folgende Bedingung␣
 ↪einfügen
def f(numbers):
    if len(numbers) == 4:
        pass # hier regulär ausführen
    else:
        raise Exception('numbers given are not 4')

# Beispiel
numbers = [1, 2, 3, 4]
print(mean(numbers))
print(mean2(numbers))
```

```
2.5
2.5
```

Übung 5:

Definieren Sie eine Funktion, welche die Standardabweichung der vier gewählten Zahlen berechnet.

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

mit N als Anzahl und $\overline{x}$ als Mittelwert der Daten.

Überprüfen Sie immer die numerischen Resultate.

[39]:
```
import math
```

```python
def mean(numbers: list) -> float:
    return sum(numbers) / len(numbers)

def standard_deviation(numbers: list) -> float:
    length = len(numbers)
    average_number = mean(numbers)
    sum = 0
    for number in numbers:      # Schleife um die Summenformel zu berechnen
        sum += (number - average_number)**2
    return math.sqrt(1/(length-1)*sum)  # berechnet die Werte in der Formel

# Beispiel
numbers = [1, 2]
print(standard_deviation(numbers))
```

```
0.7071067811865476
```

Übung 6:

Definieren Sie eine Funktion, welche die Summe aus den Quadrate der ersten n natürlichen Zahlen zurückgibt.

[44]:
```python
def sum_of_squares(n: int):
    """
    Summe der Quadrate der ersten n Zahlen, n muss positiv sein
    """
    sum = 0
    i = 1
    while i <= n:       # Schleife welche alle Quadrate der n-Zahlen addiert
        sum += i**2
        i += 1
    return sum

# Beispiel
print(sum_of_squares(5))
```

```
55
```

Übung 7:

Geben Sie ein Volumen von 75,73 m^3 an und drücken Sie es mit allen signifikanten Stellen und in wissenschaftlicher Notation aus.

[2]:
```python
x = 75.73

print(f"{x:.2} m³")
print("{:.2} m³".format(x))
```

```
7.6e+01 m³
7.6e+01 m³
```