Python

July 26, 2022

1 Python - Überblick und Aufgaben

1.1 Eckdaten

- Autoren:
  - Emrah Öztürk: emrah.oeztuerk@fhv.at, FHV, Raum V723
  - Klaus Rheinberger: klaus.rheinberger@fhv.at, FHV, Raum V721

- Version: 1
- Datum: 26. Juli 2022

1.2 Anleitung

Dieses Dokument soll Studierenden von technischen und naturwissenschaftlichen Studienrichtungen an der FHV helfen, beim Erlernen der Programmiersprache Python

- einen Überblick zu erlangen oder zu bewahren
- die ersten Erfahrungen mit Aufgaben zu festigen.

Für das eigentliche Erlernen von Python empfehlen wir das Buch "Python: Der Grundkurs" von Michael Kofler, Rheinwerk Computing, 2. Auflage, 2021. Folgende Kapitel des Buchs liefern die notwendigen Inhalte:

- Kapitel 1: Hello, World!
- Kapitel 2: Variablen
- Kapitel 3: Operatoren
- Kapitel 4: Zahlen (ohne 4.3)

- Kapitel 5: Zeichenketten (bis inklusive 5.4)
- Kapitel 7: Listen, Tupel, Sets und Dictionaries (ohne Beispiel auf Seite 118)
- Kapitel 8: Verzweigungen und Schleifen (bis inklusive 8.5)
- Kapitel 9: Funktionen (bis inklusive 9.3)
- Kapitel 20: Abschnitt 20.1 "Anaconda, IPython und Jupyter-Notebooks"

Zur Erinnerung:

- Code-Kommentare beginnen mit dem Rautezeichen `#`.
- Funktionen haben immer runde Klammern, indiziert wird mit eckigen Klammern.
- Der letzte Befehl einer Reihe von Code-Zeilen wird ausgegeben.
- Ein `;` am Code-Ende unterdrückt diese Ausgabe.
- Das Potenzieren von Zahlen wird mit `**` und nicht mit `^` implementiert.

- Im Command-Prompt oder in einer Codezelle können folgende Navigationsbefehle verwendet werden:
  - `pwd`: print current directory
  - `ls`: list files in current directory
  - `cd DIR`: change into absolute or relative directory DIR
  - `cd ..`: change to parent directory

1.3 Pakete

Diese Python-Pakete werden im Folgenden verwendet:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import optimize
```

1.4 Zeichenketten (Strings)

Der Umgang mit Zeichenketten gehört zu den Grundfertigkeiten jeder Programmiersprache. Zeichenketten werden wahlweise in einfache oder doppelte Apostrophe gesetzt. Der Zugriff auf Teilstrings erfolgt über die Angabe der Position beginnend mit dem Index 0. Strings können unter anderem konkateniert (zusammengefügt) und vervielfacht werden.

[2]:
```
my_text = "Hello, word!"
print(my_text)
```

```
Hello, word!
```

[3]:
```
# If you just want to see one variable (not only strings)
# you do not need the print function in the code cell.
# In other Python IDE like for example https://thonny.org/
# you need the print function.
my_text
```

[3]:
```
'Hello, word!'
```

[4]:
```
# only the last line of the code cell is given in the output:
my_text
my_text
```

[4]:
```
'Hello, word!'
```

[5]:
```
type(my_text) # get data type
```

[5]:
```
str
```

[6]:
```
s1 = "abc"
s2 = "efg"
```

```
print(s1 + s2 + s1) # join (=concatenate) strings
print(s1*3 + "x"*2) # join and multiply strings
```

```
abcefgabc
abcabcabcxx
```

[7]:
```
s1[1] # element of string with index 1
```

[7]:
```
'b'
```

[8]:
```
s1.upper() # string with capital letters
```

[8]:
```
'ABC'
```

[9]:
```
s1.replace("a", "x") # replace a with x in s1
```

[9]:
```
'xbc'
```

[10]:
```
s1.find("b") # get index of b in s1
```

[10]:
```
1
```

[11]:
```
name = "Tom"
age = 25

print(f"Name = {name}, Alter = {age}") # formatted printing
```

```
Name = Tom, Alter = 25
```

1.5 Zahlen

In Python können Sie problemlos mit ganzen Zahlen (integers) und Fließkommazahlen (floating point numbers) rechnen. Bei der Verwendung von Fließkommazahlen treten normalerweise Rundungsfehler auf. Eine Besonderheit von Python ist, dass es auch komplexe Zahlen unterstützt und alle grundlegenden Rechenoperationen durchführen kann.

[12]:
```
a = 3
b = 7

type(a)
```

[12]:
```
int
```

[13]:
```
a/b # division
```

[13]:
```
0.42857142857142855
```

[14]:
```
type(a/b) # the result of a division (also of integers) is a floating point␣
↪number
```

[14]: float

[15]:
```
type(10/5)
```

[15]: float

[16]:
```
2*a # multiplication
```

[16]: 6

[17]:
```
type(2*a)
```

[17]: int

[18]:
```
a + b # addition
```

[18]: 10

[19]:
```
a - b # subtraction
```

[19]: -4

[20]:
```
a//b # integer division, 3 = 0*7 + 3
```

[20]: 0

[21]:
```
a%b # remainder, 3 = 0*7 + 3
```

[21]: 3

[22]:
```
a = 2 + 3j # complex number
b = 1 - 4j # complex number

type(a)
```

[22]: complex

[23]:
```
a*b
```

[23]: (14-5j)

[24]:
```
print(a.real) # real part of a
print(a.imag) # imaginary part of a
```

```
2.0
3.0
```

1.6 Wahrheitswerte (Booleans)

Boolesche Werte können die beiden Zustände wahr (True) und falsch (False) annehmen. Dies wird zum Beispiel bei der Auswertung von Bedingungen verwendet. Außerdem können die Werte mit logischen Operatoren wie `and` und `or` verknüpft werden.

[25]:
```
a = True
b = 7 == 8 # b is false, as 7 is not equal to 8

print(b)
```

```
False
```

[26]:
```
type(a)
```

[26]:
```
bool
```

[27]:
```
not b # negate b
```

[27]:
```
True
```

[28]:
```
7 < 8 and 3 > 1.5
```

[28]:
```
True
```

[29]:
```
7 >= 8 or 4 == 4.0
```

[29]:
```
True
```

1.7 Listen

Der wichtigste Datentyp zur Speicherung einer variablen Anzahl von Elementen sind Listen. Die Ordnung der darin enthaltenen Elemente wird gespeichert. Es können an jeder beliebigen Stelle Elemente hinzugefügt werden.

[30]:
```
some_list = [1, 2.3, "a", "my python"] # make a list (order matters)
print(some_list) # print list
```

```
[1, 2.3, 'a', 'my python']
```

[31]:
```
len(some_list) # gives the length, i. e., the number of values in the list
```

[31]:
```
4
```

[32]:
```
some_list.append("new item") # add new item to list at the end
print(some_list)
```

```
[1, 2.3, 'a', 'my python', 'new item']
```

[33]:
```
some_list.extend([3, 1]) # add new items to the list (same result as + operator)
some_list
```

[33]:
```
[1, 2.3, 'a', 'my python', 'new item', 3, 1]
```

[34]:
```
some_list.count(1) # count value 1 in list
```

[34]:
```
2
```

[35]:
```
# first 2 items:
print(some_list[:2])

# items with index 1, 2 and 3:
print(some_list[1:4])

# last two items in the list:
print(some_list[-2:])
```

```
[1, 2.3]
[2.3, 'a', 'my python']
[3, 1]
```

[36]:
```
print(some_list)

del some_list[0:3] # delete list elements with index 0, 1 and 2
print(some_list)
```

```
[1, 2.3, 'a', 'my python', 'new item', 3, 1]
['my python', 'new item', 3, 1]
```

[37]:
```
some_list.clear() # delete all list elements
print(some_list)
```

```
[]
```

[38]:
```
some_list = [1, 3, 2]
print(some_list.index(3)) # get index of value 3 in list
```

```
1
```

[39]:
```
some_list.insert(1, 10) # insert value 10 in position 1
print(some_list)
```

```
[1, 10, 3, 2]
```

[40]:
```
max(some_list) # get max value in list
```

[40]:
```
10
```

[41]:
```
min(some_list) # get min value in list
```

[41]:
```
1
```

[42]:
```
some_list.reverse() # reverse list

print(some_list)
```

```
[2, 3, 10, 1]
```

[43]:
```
print(sorted(some_list)) # sort elements without changing list

print(some_list)
```

```
[1, 2, 3, 10]
[2, 3, 10, 1]
```

[44]:
```
some_list.sort() # sort elements in list with changing the list
print(some_list)
```

```
[1, 2, 3, 10]
```

[45]:
```
some_list.remove(10) # remove value 10 from list
print(some_list)
```

```
[1, 2, 3]
```

[46]:
```
some_list_2 = [i for i in range(4, 6)] # list comprehension
print(some_list_2)
```

```
[4, 5]
```

[47]:
```
print(list(zip(some_list, some_list_2))) # get combined values of two lists
```

```
[(1, 4), (2, 5)]
```

[48]:
```
print(1 in some_list) # check if one is included in the list
```

```
True
```

1.8 Tupel

Ein Tupel ist, vereinfacht gesagt, eine unveränderliche Liste mit wenigen Elementen. Tupel sind von ihren Möglichkeiten wesentlich simpler als Listen, intern aber auch mit weniger Overhead verbunden. Sie werden oft verwendet, um mehrere zusammengehörende Daten an Funktionen zu übergeben oder als Ergebnis zurückzugeben.

[49]:
```
t = (1, 2, "a") # defines a tuple
print(t)
```

```
(1, 2, 'a')
```

[50]:
```
t[0] # get first element in t
```

[50]:
```
1
```

[51]:
```
# t[0] = 2 # this code gives an error (tuples are not changeable)
```

[52]:
```
("a", "b", "c") == (1, 2, 3) # check if all elements are the same
```

[52]:
```
False
```

[53]:
```
a = 17
b = 23
(a, b) = (b, a) # swap elements
print(a)
print(b)
```

```
23
17
```

1.9 Set (Menge)

Ein Set ist eine Menge von ungeordneten Elementen. Sie enthält also keine Doppelgänger.

[54]:
```
s = {1, 2, 3, 3} # defines a set (order does not matter)
print(s)
```

```
{1, 2, 3}
```

[55]:
```
s.add(4) # adds 4 to the set
print(s)
```

```
{1, 2, 3, 4}
```

[56]:
```
s.remove(1) # removes the element 1
print(s)
```

```
{2, 3, 4}
```

[57]:
```
s.clear() # removes all elements
print(s)
```

```
set()
```

[58]:
```
x = set("abcdef")
y = set("efgh")

print(x)
print(y)
print(x | y) # union of x and y ((elements in x or in y)
```

```
{'b', 'f', 'a', 'd', 'c', 'e'}
{'e', 'f', 'g', 'h'}
{'b', 'g', 'f', 'a', 'd', 'h', 'c', 'e'}
```

[59]:
```
print(x - y) # set difference of x and y
```

```
{'a', 'c', 'b', 'd'}
```

[60]:
```
print(x & y) # intersection of x and y
```

```
{'e', 'f'}
```

[61]:
```
print(x ^ y) # xor of x and y (elements in x or in y but not in the␣
 ↪intersection of x and y)
```

```
{'b', 'g', 'a', 'd', 'h', 'c'}
```

1.10 Dictionaries

In Dictionaries können Sie Wertepaare (Key-Value-Paare) speichern. Der wesentliche Unterschied im Vergleich zu Listen besteht darin, dass der Zugriff auf einzelne Werte über den Schlüssel (Key) erfolgt, nicht über einen durchlaufenden Index.

[62]:
```
# define a dictionary
battery = {"x_max": 7.6, "x_min": -7.6, "S_max": 13.5, "S_0": 10.5, "alpha": 0.
 ↪95}

print(battery)
```

```
{'x_max': 7.6, 'x_min': -7.6, 'S_max': 13.5, 'S_0': 10.5, 'alpha': 0.95}
```

[63]:
```
battery["x_max"] # get value with key x_max
```

[63]:
```
7.6
```

[64]:
```
battery.keys() # get all keys
```

[64]:
```
dict_keys(['x_max', 'x_min', 'S_max', 'S_0', 'alpha'])
```

[65]:
```
battery.values() # get all values
```

[65]:
```
dict_values([7.6, -7.6, 13.5, 10.5, 0.95])
```

[66]:
```
battery.items() # get combinations of keys and values
```

[66]:
```
dict_items([('x_max', 7.6), ('x_min', -7.6), ('S_max', 13.5), ('S_0', 10.5),
('alpha', 0.95)])
```

[67]:
```
print("x_max" in battery) # checks whether "x_max" is a key in x
```

```
True
```

[68]:
```
battery["eta"] = 0.95 # add new key eta with value 0.95
print(battery)
```

```
{'x_max': 7.6, 'x_min': -7.6, 'S_max': 13.5, 'S_0': 10.5, 'alpha': 0.95, 'eta':
0.95}
```

[69]:
```
del battery["eta"] # delete key eta
print(battery)
```

```
{'x_max': 7.6, 'x_min': -7.6, 'S_max': 13.5, 'S_0': 10.5, 'alpha': 0.95}
```

1.11 If-Elif-Else Abfrage

Die Syntax lässt sich wie folgt zusammenfassen:

```
if Bedingung1:
    Block1
elif Bedingung2:
    Block2
elif Bedingung3:
    Block3
else:
    Block4
```

Es sind beliebig viele elif-Bedingungen erlaubt, auch null. Der else-Block ist optional.

[70]:
```
x = 41

if x == 42:
    print("Hurra!")
else:
    print("Hmm!")
```

```
Hmm!
```

[71]:
```
x = 42

if x == 42:
    print("Hurra!")
```

```
Hurra!
```

[72]:
```
n = 3
s = "even" if n%2 == 0 else "odd" # shorthand if notation
print(f"{n} is {s}")
```

```
3 is odd
```

1.12 For-Schleife

Schleifen werden in Python zumeist mit `for var in elemente` gebildet. Die Schleifenvariable nimmt dabei der Reihe nach jedes der angegebenen Elemente an.

```
for var in elemente:
    Anweisungen
```

[73]:
```
for i in range(100, 0, -10): # use of range function in for loop
    print(i, end=" ")
```

```
100 90 80 70 60 50 40 30 20 10
```

[74]:
```
lst = ["a", "b", "c"]
cnt = 0
for item in lst:
    print(cnt, item)
    cnt += 1 # increase counter by 1
```

```
0 a
1 b
2 c
```

[75]:
```
for cnt, item in enumerate(lst): # use of enumerate function in for loop
    print(cnt, item)
```

```
0 a
1 b
2 c
```

[76]:
```
lst = [1, 2, 3, 4]
print([x**2 for x in lst]) # use of list comprehension
```

```
[1, 4, 9, 16]
```

[77]:
```
battery = {"x_max": 7.6,
           "x_min": -7.6,
           "S_max": 13.5,
           "S_0": 10.5,
           "alpha": 0.95}

for key, value in battery.items(): # loop through the dictionary
    print(f"{key}: {value}")
```

```
x_max: 7.6
x_min: -7.6
S_max: 13.5
S_0: 10.5
alpha: 0.95
```

[78]:
```
# use of generator expression (faster than with for loop or list comprehension):
print(sum(x**2 for x in range(100)))
```

```
328350
```

[79]:
```
for i in range(10): # use of break in a for loop
    if i == 2:
        break
    else:
        print(i, end=" ")
```

```
0 1
```

[80]:
```
for i in range(10): # use of continue in a for loop
    if i == 2:
        continue
    else:
        print(i, end=" ")
```

```
0 1 3 4 5 6 7 8 9
```

1.13 While-Schleife

Als Alternative zu `for` können Sie Schleifen mit `while` formulieren. Die eingerückten Anweisungen werden dann so lange ausgeführt, wie die Bedingung erfüllt ist. Die Schlüsselwörter `break` und `continue` funktionieren wie bei der `for`-Schleife.

```
while Bedingung:
    Anweisungen
```

[81]:
```
i = 1

while i < 5: # while loop
    print(i, end= " ")
```

```
    i += 1 # increase i by 1. Caution: If this expression is omitted, the loop␣
 ↪will continue forever!
```

```
1 2 3 4
```

1.14 Funktionen

Funktionen können den Code besser und übersichtlicher strukturieren. In vielen Fällen können Funktionen auch Redundanz im Code vermeiden, z. B. wenn Sie eine bestimmte Aufgabe an mehreren Stellen in Ihrem Code durchführen müssen.

```
def funktionsname(para1, para2, para3):
    code
    mehr code
    return rückgabe
```

Die Verwendung von `return` ist optional. Funtionen können auch vorzeitig mit `return` verlassen werden.

[82]:
```
def f1(): # defines a function
    y = 5
    print(y)

f1()
# print(y) # this code gives an error as y is only defined locally in the␣
 ↪function body
```

```
5
```

[83]:
```
def f1(): # defines a function
    y = 5
    return y

y = f1()
print(y)
```

```
5
```

[84]:
```
# Caution: changing an element of a list affects the list also outside the␣
 ↪function
def f3(x):
    x.append(3)
    print(x)

x = [1, 2]
f3(x)
print(x)
```

```
[1, 2, 3]
[1, 2, 3]
```

[85]:
```
def f4(a, b, c=-1, d=0): # function with optional arguments
    print(a, b, c, d)

f4(6, 7, 8, 9)
f4(6, 7)
```

```
6 7 8 9
6 7 -1 0
```

[86]:
```
def f5(a, b, *c): # defines a function with a variable amount of arguments
    print(a, b, c)

f5(1, 2)
f5(1, 2, 3)
f5(1, 2, 3, 4, 5, 6)
```

```
1 2 ()
1 2 (3,)
1 2 (3, 4, 5, 6)
```

[87]:
```
def f6(a, b, c):
    print("a =", a, ", b =", b,", c =", c, sep=" ")

f6(1, 2, 3)        # order of variables matters

f6(c=3, b=1, a=-7) # oder of variables does not mater
```

```
a = 1 , b = 2 , c = 3
a = -7 , b = 1 , c = 3
```

[88]:
```
# example for recursive function:

def f7(n):
    if n <= 1:
        return 1
    else:
        return n*f7(n-1)

print(f7(5))
```

```
120
```

[89]:
```
def f8(x, y):
    return (x+1)*(y+1)
```

```
f9 = lambda x, y: (x+1)*(y+1) # definition of function with lambda operator

print(f8(2, 3))
print(f9(2, 3))
```

```
12
12
```

[90]:
```
# map list elements with defined function (square):
lst = [1, 2, 3, 4, 5]
list(map(lambda x: x**2, lst))
```

[90]:
```
[1, 4, 9, 16, 25]
```

[91]:
```
# filter list elements with defined function:
list(filter(lambda x: x % 2 == 0, lst))
```

[91]:
```
[2, 4]
```

1.15 Try-Except

Python gibt Ihnen die Möglichkeit, Ihren Code mit `try-except` abzusichern. Damit können Sie Ihr Programm selbst dann fortsetzten, wenn eine Exception auftritt.

```
try:
    # fehleranfälliger Code
except anError:
    # Reaktion auf den bestimmten Fehler vom Typ anError
else:
    # Wird ausgeführt, wenn kein Fehler aufgetreten ist.
finally:
    # Wird immer ausgeführt, egal, ob ein Fehler aufgetreten ist oder nicht.
```

Umgekehrt kann es bei der Programmierung eigener Funktionen auch zweckmäßig sein, eigene Fehler auszulösen: Sie weisen damit andere Programmierer darauf hin bzw. erinnern sich selbst daran, wie die Funktion zu nutzen ist.

[92]:
```
if 'z' in locals(): # check if variable z is defined.
    del z # delete if defined
```

[93]:
```
z = 1 # try again after deleting z and commenting this line out

try:
    print(z)
except NameError:
    print("ERROR: Variable z is not defined!")
except:
    print("ERROR: Something else went wrong!")
```

```
else:
    z = 12
finally:
    if 'z' in locals():
        print(z)
    print('finished.')
```

```
1
12
finished.
```

[94]:
```
def f10(x, y):
    if x < 0 or y < 0:
        raise ValueError("Use positive parameters!") # defines excepetion
    return x*y

print(f10(1, 2))
```

```
2
```

[95]:
```
# f10(-1,2) # This code raises a ValueError
```

1.16 NumPy

Das Paket Numpy stellt Funktionen zur Erzeugung und Verarbeitung von (mehrdimensionalen) Arrays zur Verfügung. Damit können Sie mit Vektoren und Matrizen effizient rechnen. Eine ausführliche Dokumentation finden Sie auf der Seite numpy.org.

[96]:
```
np.zeros([2, 3]) # gives an 2x3 array of zeros
```

[96]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

[97]:
```
np.ones([3, 4]) # gives an 3x4 array of ones
```

[97]:
```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

[98]:
```
np.eye(2) # gives the 2x2 identity matrix
```

[98]:
```
array([[1., 0.],
       [0., 1.]])
```

[99]:
```
np.random.rand(2, 2) # gives a randomly generated 2x2 array
```

[99]:
```
array([[0.03871333, 0.40632759],
       [0.00687199, 0.96620115]])
```

[100]:
```
A = np.array([[1, 2, 3],
              [4, 5, 6]]) # defines array

print(np.shape(A))
```

```
(2, 3)
```

[101]:
```
x = np.array([4, 3])

np.linalg.norm(x, 2) # euclidean distance of x (other norms are possible by␣
 ↪changing the second argument)
```

[101]:
```
5.0
```

[102]:
```
A = np.array([[1, 2],
              [3, 4]])

np.linalg.inv(A) # gives the inverse of A (AA^(-1) = A^(-1)A = I)
```

[102]:
```
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

[103]:
```
A@x # vector matrix multiplication
```

[103]:
```
array([10, 24])
```

[104]:
```
2*x # elementwise multiplication of x with 2
```

[104]:
```
array([8, 6])
```

[105]:
```
x*x # elementwise multiplication
```

[105]:
```
array([16,  9])
```

[106]:
```
A + 1 # generates an array of ones with same dimensions as A and adds this to A
```

[106]:
```
array([[2, 3],
       [4, 5]])
```

[107]:
```
A + A # adds array A to A
```

[107]:
```
array([[2, 4],
       [6, 8]])
```

[108]:
```
A.T # gives A transposed
```

[108]:
```
array([[1, 3],
       [2, 4]])
```

[109]:
```
A[:, 0] # gives first column of A
```

[109]:
```
array([1, 3])
```

[110]:
```
A[1, :] # gives second row of A
```

[110]:
```
array([3, 4])
```

[111]:
```
A.ravel() # returns the flattened array
```

[111]:
```
array([1, 2, 3, 4])
```

[112]:
```
# vector with values from 1 to 5 (included) with 10 values:
np.linspace(1, 5, 10)
```

[112]:
```
array([1.        , 1.44444444, 1.88888889, 2.33333333, 2.77777778,
       3.22222222, 3.66666667, 4.11111111, 4.55555556, 5.        ])
```

[113]:
```
# vector with values form 1 to 5 (not included) in 0.5 steps:
np.arange(1, 5, 0.5)
```

[113]:
```
array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5])
```

[114]:
```
A_temp = np.ones([len(A), 1])
np.concatenate((A, A_temp), axis = 1) # appends A_temp to A as column
```

[114]:
```
array([[1., 2., 1.],
       [3., 4., 1.]])
```

[115]:
```
from scipy import misc # array as image with 3 axes

img = misc.face()
print(img.ndim) # get number of axes
```

```
3
```

[116]:
```
plt.imshow(img); # show image
```

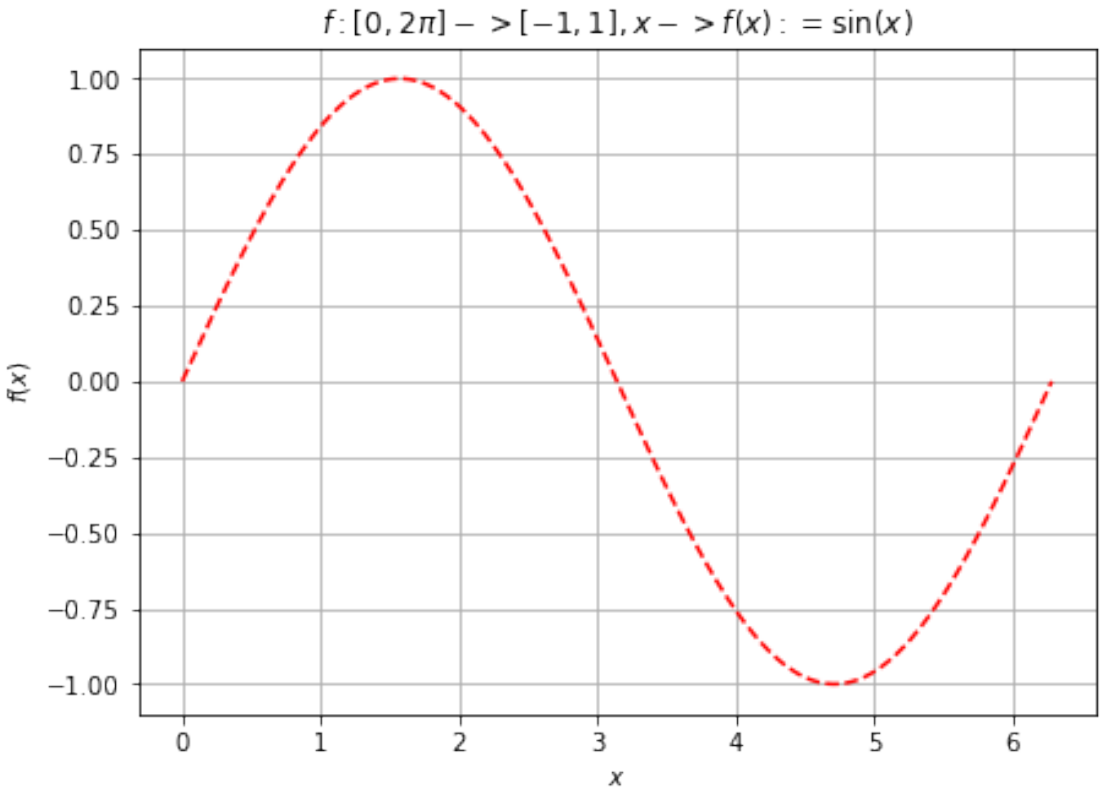
[117]:
```
np.shape(img) # get shape of image
```

[117]:
```
(768, 1024, 3)
```

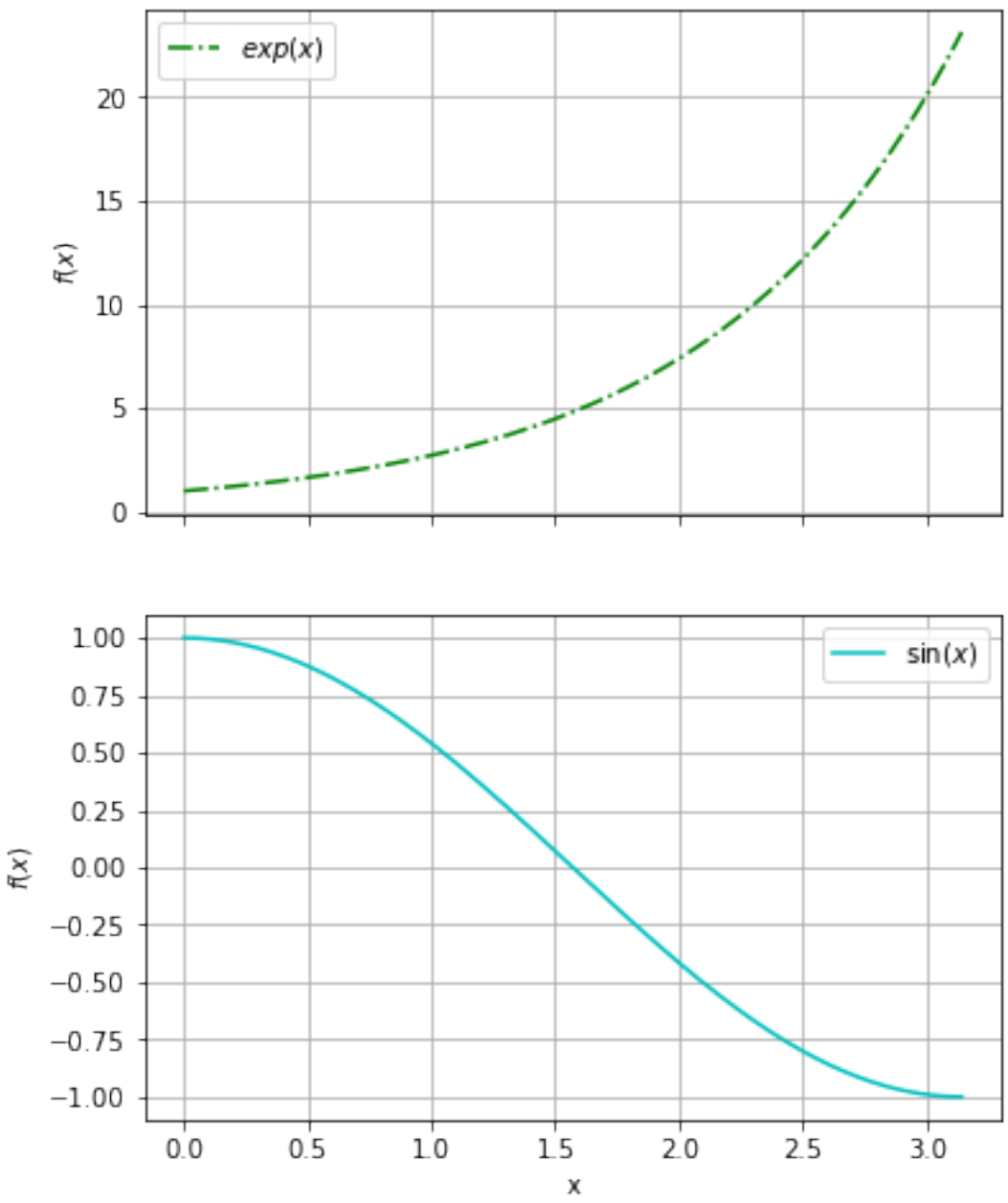
1.17 Matplotlib

Matplotlib ist eine Bibliothek mit zahlreichen Funktionen zur visuellen Darstellung von Daten. Eine ausführliche Dokumentation finden Sie unter matplotlib.org.

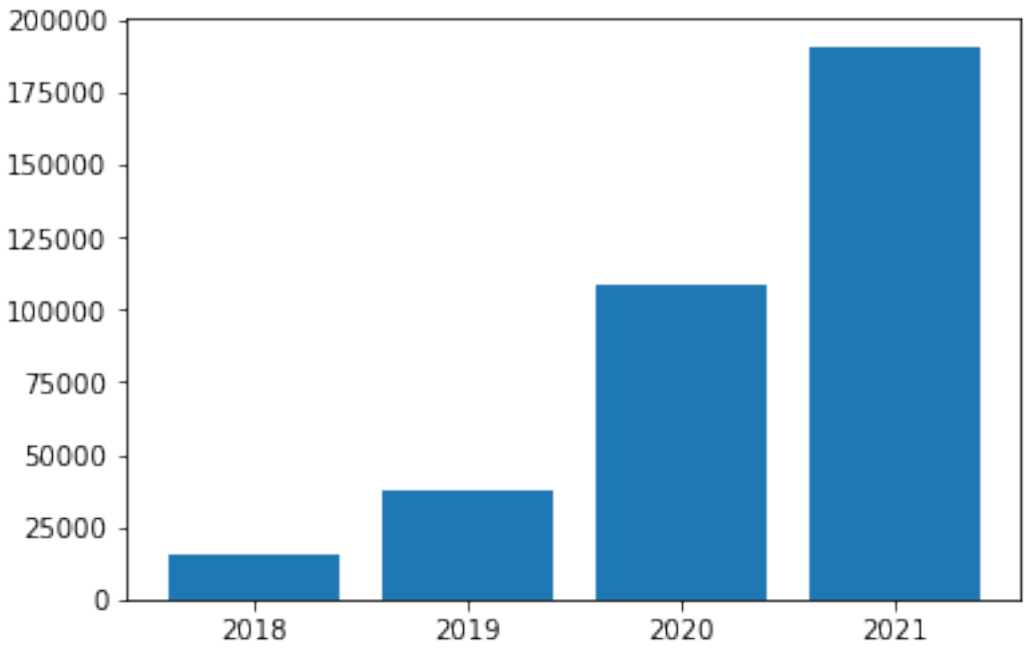
[118]:
```
x = np.linspace(0, 2*np.pi, 1000) # create x values
y = np.sin(x) # calculate y values

plt.figure(figsize=(7,5))
plt.plot(x, y, linestyle = "--", color = "r") # plot values with dashed (--)␣
 ↪linestyle and red (r) color
plt.grid(True) # make grid
plt.xlabel("$x$") # label x axis
plt.ylabel("$f(x)$") # label y axis
plt.title("$f: [0,2\pi] -> [-1,1], x -> f(x) := \sin(x)$"); # make title
```



[119]:
```
x = np.linspace(0, np.pi, 1000)
y1 = np.exp(x)
y2 = np.cos(x)

fig, ax = plt.subplots(2, 1, sharex=True, figsize=(6, 8)) # make subplots with␣
 ↪two rows, one column and shared x-axis
ax[0].plot(x, y1, linestyle = "-.", color = "g", label="$exp(x)$") # plot first␣
 ↪function (exp(x))
ax[0].grid(True)
ax[0].legend() # make legend (first row)
ax[0].set_ylabel("$f(x)$") # label y axis (first row)

ax[1].plot(x, y2, linestyle = "-", color = "c" ,label="$\sin(x)$") # plot␣
 ↪second function (sin(x))
ax[1].grid(True)
ax[1].legend() # make legend (second row)

ax[1].set_xlabel("x") # label x axis
ax[1].set_ylabel("$f(x)$"); # label y axis (second row)
```



[120]:
```
x = ["2018", "2019", "2020", "2021"] # years
y = [15510, 37850, 108205, 190727]   # number of new electric cars sold in the␣
 ↪UK by year

plt.bar(x, y); # make barchart
```



1.18 SciPy

SciPy ist eine Bibliothek mit zahlreichen mathematischen Funktionen, die vor allem im naturwissenschaftlichen Bereich benötigt werden. Zu den Themenfeldern zählen:

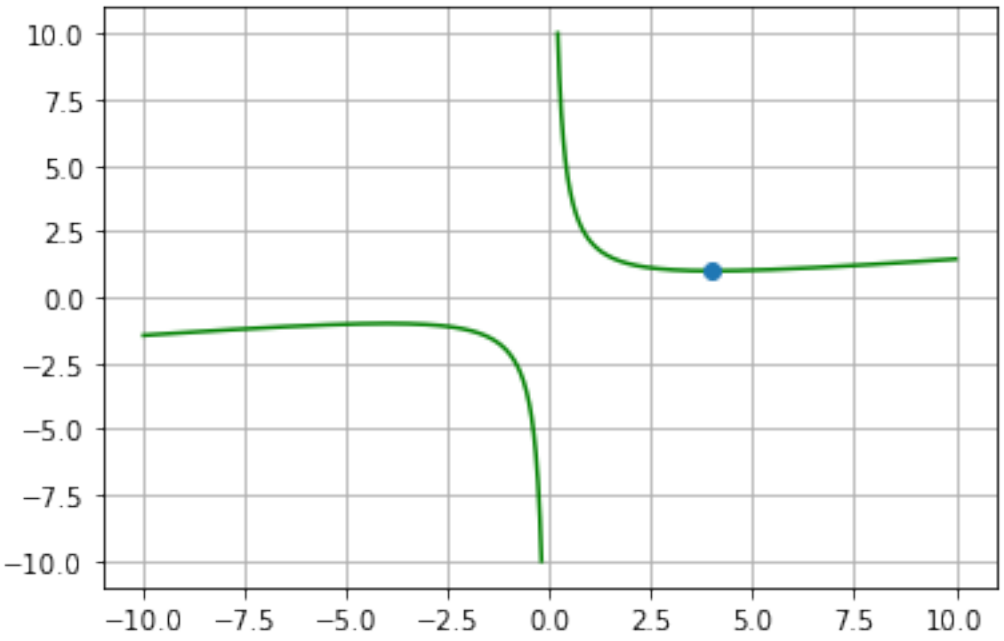
- Numerische Optimierung, Interpolation und Integration
- Statistik
- Fourier Transformationen
- Bild- und Signalverarbeitung

Eine ausführliche Dokumentation finden Sie auf der Seite scipy.org.

[121]:
```
f11 = lambda x: x/8 + 2/x
result = optimize.minimize(f11, x0 = 1) # optimization without constraints
print(result)

x = np.linspace(0.2, 10, 1000)
plt.plot(x, f11(x), "g-", result.x, result.fun, "o") # plot results
x = np.linspace(-0.2, -10, 1000)
plt.plot(x, f11(x), "g-")
plt.grid(True)
```

```
      fun: 1.0000000006051475
 hess_inv: array([[15.5937316]])
      jac: array([-8.70227814e-06])
  message: 'Optimization terminated successfully.'
     nfev: 18
```

```
     nit: 8
    njev: 9
  status: 0
 success: True
       x: array([3.99986085])
```



[122]:
```
# linear programming example:

c = np.array([-1, 4])
A = np.array([[-3, 1],
              [ 1, 2]])
b = np.array([6, 4])
x0_bounds = (None, None)
x1_bounds = (-3, None)
res = optimize.linprog(c, A_ub=A, b_ub=b, bounds=(x0_bounds, x1_bounds))
print(res)
```

```
     con: array([], dtype=float64)
     fun: -21.99999984082497
 message: 'Optimization terminated successfully.'
     nit: 6
   slack: array([3.89999997e+01, 8.46872172e-08])
  status: 0
 success: True
       x: array([ 9.99999989, -2.99999999])
```

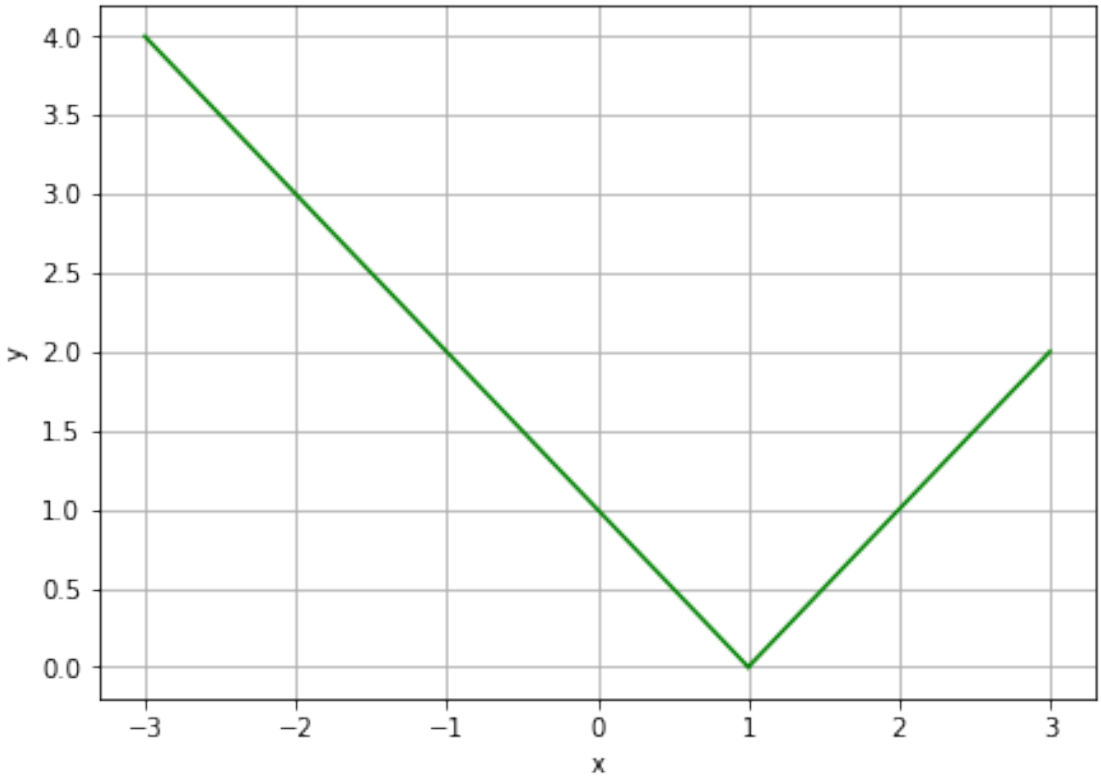
2 Aufgaben

2.1 Aufgabe 1

Erstellen Sie einen Plot der Funktion $|1 - x|$ im Bereich $x \in [-3, 3]$ inklusive Achsenbeschriftung, ohne die Funktionen `np.abs()` oder `abs()` zu verwenden.

Lösung:

[123]:
```
x_1 = np.linspace(-3, 1)
x_2 = np.linspace( 1, 3)

def f_1(x_1):
    return 1 - x_1

def f_2(x_2):
    return -(1 - x_2)

y_1 = f_1(x_1)
y_2 = f_2(x_2)

plt.figure(figsize=(7,5))
plt.plot(x_1,y_1, color = "g")
plt.plot(x_2,y_2, color = "g")
plt.xlabel("x")
plt.ylabel("y")
plt.grid(True)
```



2.2 Aufgabe 2

Die Flugbahn eines Geschosses laute (der Luftwidertand bleibt unberücksichtigt) $y(x) = -x^2 + 5x + x_0$,
wobei x_0 die x-Koordinate des Startpunktes ist. Das Geschoss erreicht die Erdoberfläche ($y = 0$) bei
$x = \frac{5}{2} + \sqrt{\frac{25}{4} + x_0}$.

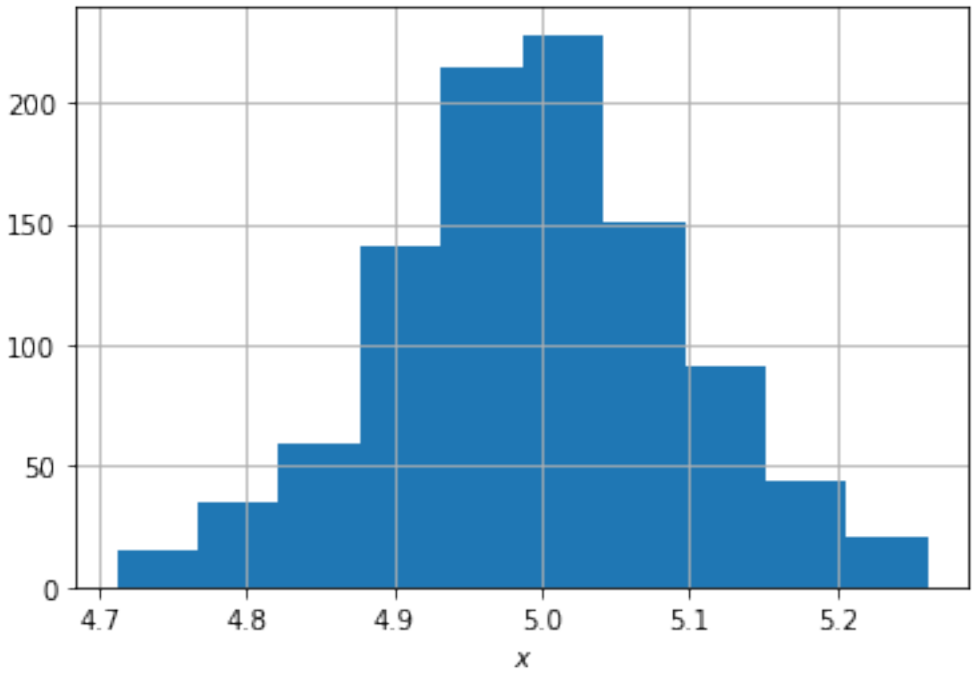
1. Erstellen Sie mit dem numpy Befehl `random.normal` 1000 normalverteilte x_0 Werte mit Erwartungswert 0 und Varianz 0.5.
2. Implementieren Sie die Funktion, welche den Ort des Aufpralls ausgibt (zweite Gleichung) mit x_0 als Übergabeparameter.
3. Werten Sie die Funktion an den zufällig erzeugten x_0 Werten aus, und erstellen Sie ein Histogramm Ihrer Ergebnisse.
4. Berechnen Sie Mittelwert und Standardabweichung der Aufprallstelle.

Lösung:

[124]:
```
mu    = 0
sigma = 0.5
x_0 = np.random.normal(mu, sigma, 1000)

def f(x_0):
```

```
    return 5/2 + np.sqrt(25/4 + x_0)

x = f(x_0)
plt.hist(x)
plt.grid()
plt.xlabel("$x$")

print(f"Mittelwert: {np.mean(x)}")
print(f"Standartabweichung: {np.std(x)}")
```

```
Mittelwert: 4.9957107936383744
Standartabweichung: 0.09984842564400884
```



2.3 Aufgabe 3

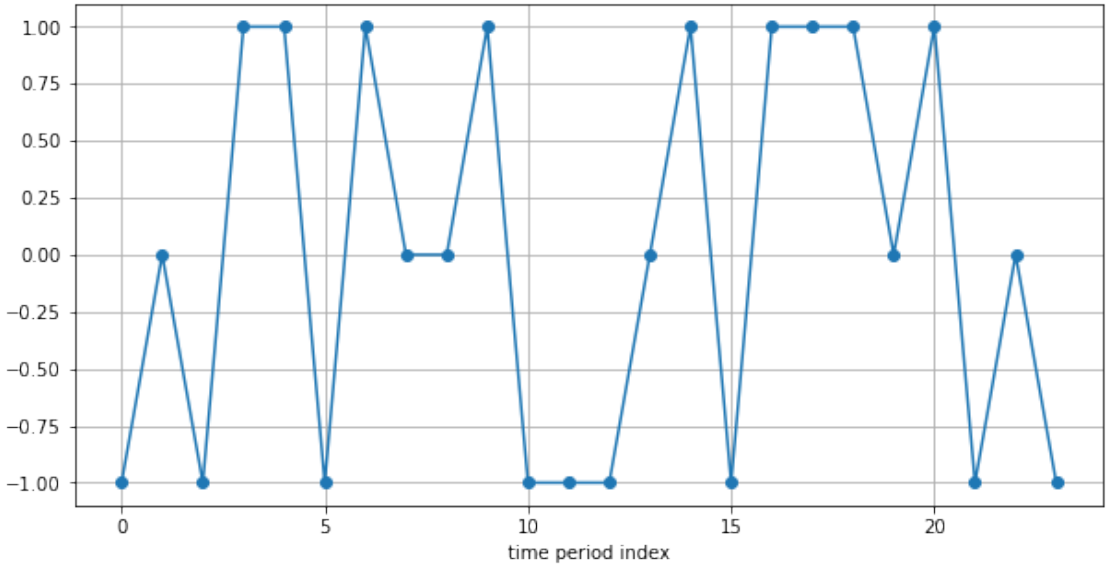
Stellen Sie das Lade-Entlade-Verhalten einer Batterie grafisch dar. Erzeugen Sie dazu einen Vektor x der Länge 24 mit Zufallswerten $\{-1, 0, 1\}$, wobei -1 für das Entladen und 1 für das Laden der Batterie steht. Verwenden Sie den random seed 2 für die Zufallszahlen.

Optional: Verwenden Sie den Tag 2022-04-27 von 00:00 bis einschließlich 23:00 für die x-Achse mittels pandas.

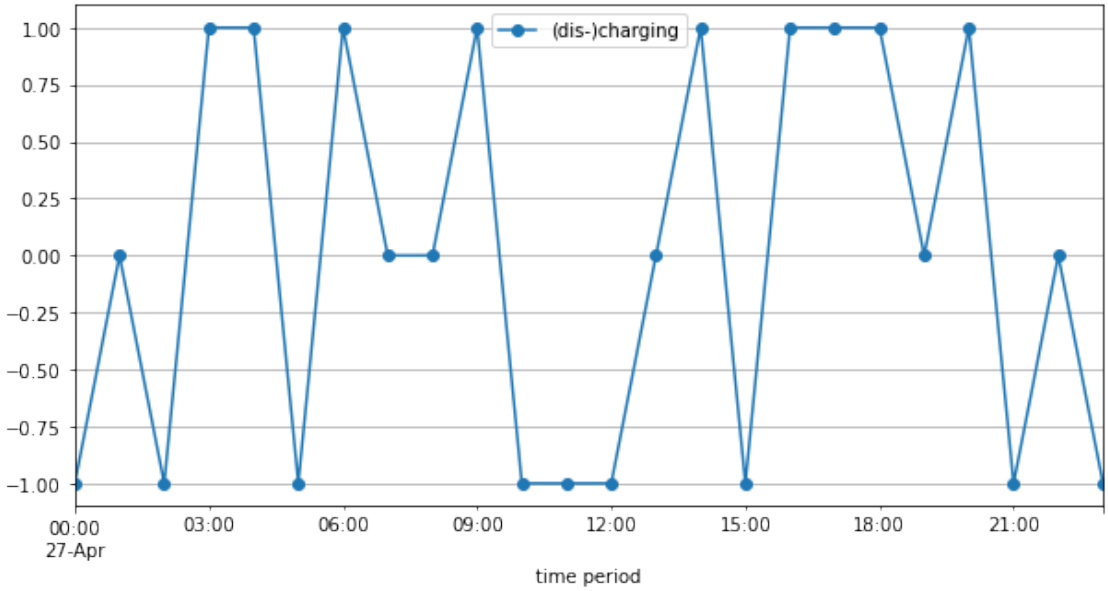
Lösung:

[125]:
```
np.random.seed(2)
x = np.random.randint(-1, 2, 24)
```

```
plt.figure(figsize=(10,5))
plt.plot(x, 'o-')
plt.xlabel("time period index")
plt.grid()
```



[126]:
```
idx = pd.date_range("2022-04-27", periods=24, freq="H")
ts = pd.DataFrame(x, columns = ["(dis-)charging"], index=idx)
ts.plot(style="o-", figsize=(10,5), xlabel="time period", grid=True);
```

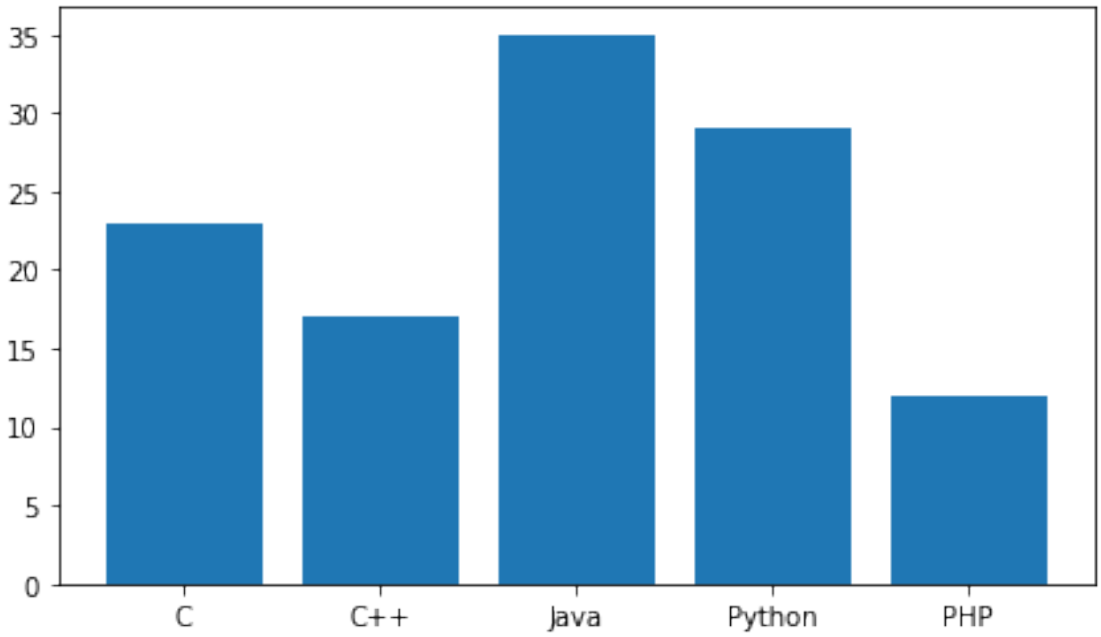


2.4 Aufgabe 4

Sie machen eine Umfrage, welche Programmiersprachen häufig verwendet werden. Die Programmiersprachen C, C++, Java, Python und PHP stehen zur Auswahl. Sie erhalten die Zahlen $(23, 17, 35, 29, 12)$ als Antwort. Stellen Sie den Vektor entsprechend grafisch dar. Welche Darstellungsform könnte hier gewählt werden?

Lösung:

[127]:
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
frequencies = [23, 17, 35, 29, 12]

plt.figure(figsize=(7,4))
plt.bar(langs, frequencies);
```



2.5 Aufgabe 5

Gehen Sie das folgende Programm gedanklich durch. Was passiert in jeder Zeile. Versuchen Sie, die Ausgabe des Programms ohne Programmierung zu finden.

```
wordlist1 = ['Python', 'ist', 'cool']
wordlist2 = wordlist1
wordlist1.insert(2,'total')
print(wordlist1)
print(wordlist2)
```

Lösung:

Listen sind ein `mutable` Datentyp! Verweist eine Variable auf ein Objekt vom Typ mutable, und ändert sich dieses Objekt, dann ändert sich auch der Inhalt der Liste.

[128]:
```
wordlist1 = ["Pyton", "ist", "cool"]
wordlist2 = wordlist1
wordlist1.insert(2,"total")
print(wordlist1)
print(wordlist2)
```

```
['Pyton', 'ist', 'total', 'cool']
['Pyton', 'ist', 'total', 'cool']
```

2.6 Aufgabe 6

Welchen Wert gibt das folgende Programm aus? Versuchen Sie, die Ausgabe des Programms ohne Programmierung zu finden.

```
a = 'abcde'
b = a
a = a + 'fg'
print(a)
print(b)
```

Lösung:

Strings sind ein `immutable` Datentyp!

[129]:
```
a = "abcde"
b = a
a = a + 'fg'
print(a)
print(b)
```

```
abcdefg
abcde
```

2.7 Aufgabe 7

Finden Sie eine Erkärung für die Ausgabe im folgenden Code. Wie könnte eine Lösung aussehen?

[130]:
```
a = 0.7
b = 0.9
c = a + 0.1
d = b - 0.1
print(c == d)
```

```
False
```

Lösung:

[131]:
```
# mögliche Lösung 1:
a = 0.7
b = 0.9
c = a + 0.1
d = b - 0.1

e = np.finfo(float).eps
print(abs(c - d) <= e) # Überprüfen auf Gleichheit

# mögliche Lösung 2:
from decimal import Decimal # Berechnung mit decimal ist aber langsam!
a = Decimal("0.7")
b = Decimal("0.9")
c = a + Decimal("0.1")
d = b - Decimal("0.1")
print(c == d)
print(d - c)
```

```
True
True
0.0
```

2.8 Aufgabe 8

Sie wollen reproduzierbare Ergbnisse eines Experiments und verwenden dazu den random seed 1. Was stimmt mit dem unteren Code nicht? Wie kann er gegebenfalls korrigiert werden?

[132]:
```
np.random.seed(1)
print(np.random.uniform(1, 2))

import random

random.seed(1)
print(random.uniform(1, 2))
```

```
1.4170220047025741
1.134364244112401
```

Lösung:

Die Pakete numpy.random und random implementieren Zufallszahlen unterschiedlich. Um konsistente Ergebnisse für einen gewählten seed zu erhalten, muss man bei einem Paket bleiben.

[133]:
```
# Möglichkeit 1:
np.random.seed(1)
print(np.random.uniform(1,2))
np.random.seed(1)
print(np.random.uniform(1,2))
```

```
# Möglichkeit 2:
random.seed(1)
print(random.uniform(1,2))
random.seed(1)
print(random.uniform(1,2))
```

```
1.4170220047025741
1.4170220047025741
1.134364244112401
1.134364244112401
```

2.9 Aufgabe 9

Entfernen Sie die Doppelgänger aus einer Liste von Zahlen, z. B. aus [1, 2, 3, 2, 7, 3, 9]. Die Ergebnisliste soll aufsteigend sortiert sein.

Lösung:

[134]:
```
lst = [1, 2, 3, 2, 7, 3, 9]
lst = list(set(lst))
lst.sort()
print(lst)
```

```
[1, 2, 3, 7, 9]
```

2.10 Aufgabe 10

Extrahieren Sie aus der folgenden Zeichenkette die Zeichen zwischen den eckigen Klammern:

```
bla [wichtig] mehr bla
```

Lösung:

[135]:
```
s = "bla [wichtig] mehr bla"
start = s.find("[") + 1
end = s.find("]")
print(s[start:end])
```

```
wichtig
```

2.11 Aufgabe 11

Entscheiden Sie für jede Codezeile, ob der Boolesche Ausdruck wahr oder falsch ist, ohne das Beispiel in die Python-Konsole einzugeben. Prüfen Sie die Antwort anschliessend mit der Konsole.

```
3 > 4
4 != 5
"Hallo" == "Hallo Welt"
not not 5 == 5
not 3 >= 4 and not 4 >= 5
```

Lösung:

[136]:
```
print(3 > 4)
print("Hallo" == "Hallo Welt")
print(not not 5 == 5) # 5 == 5 is true, not not true is ture
print(not 3 >= 4 and not 4 >= 5)
```

```
False
False
True
True
```