

# Grüne Software-Erstellung

Informatica Feminale 2022

Mercedes-Benz Tech Innovation





Past

Mercedes-Benz Tech Innovation

Now

# Wir gestalten die digitale Zukunft von Mercedes-Benz

Wir sind professionelle **Software-Entwickler**, **produktorientierte** Projektmanager, strategische Partner, kreative Visionäre, zuverlässige Impulsgeber, Change Enabler und Tech Innovators. Wir sind **stolzer Teil der Mercedes-Benz Familie** und eine **tragende Säule** innerhalb der IT.

Als **hundertprozentige** Tochtergesellschaft entwickeln wir technologische Innovationen, digitale Produkte und zukunftsweisende Software-Lösungen exklusiv für Mercedes-Benz. Dabei treiben wir die Transformation in Richtung „100% digital“. Wir setzen neue Standards für **Car Connectivity** und in der **Digitalisierung** von **Fahrzeug** und **Vertrieb**.



# Wer bin ich?

## Dr. Sabine Gillner

Produktmanagement Mercedes Benz Tech Innovation

Developer Experience

FOSS

Unternehmensbetreuerin

MPI biologische Kybernetik, Universität Tübingen



# Agenda

## 1. Block: 09:00 – 10:30

- I. Energieverbrauch weltweit
- II. Energieverbrauch durch ITK
- III. Prinzipien zur Optimierung von python code

## 2. Block: 11:00 – 12:30

- I. Ein Blick auf Maßeinheiten zur Messung des Energieverbrauches
- II. Das Paket numpy
- III. Visualisierung mit codecarbon[viz]

## 3. Block: 13:30 – 15:00

- I. Nachhaltige Programmiersprachen
- II. Binary tree – Messung
- III. Optimierung Teil 2



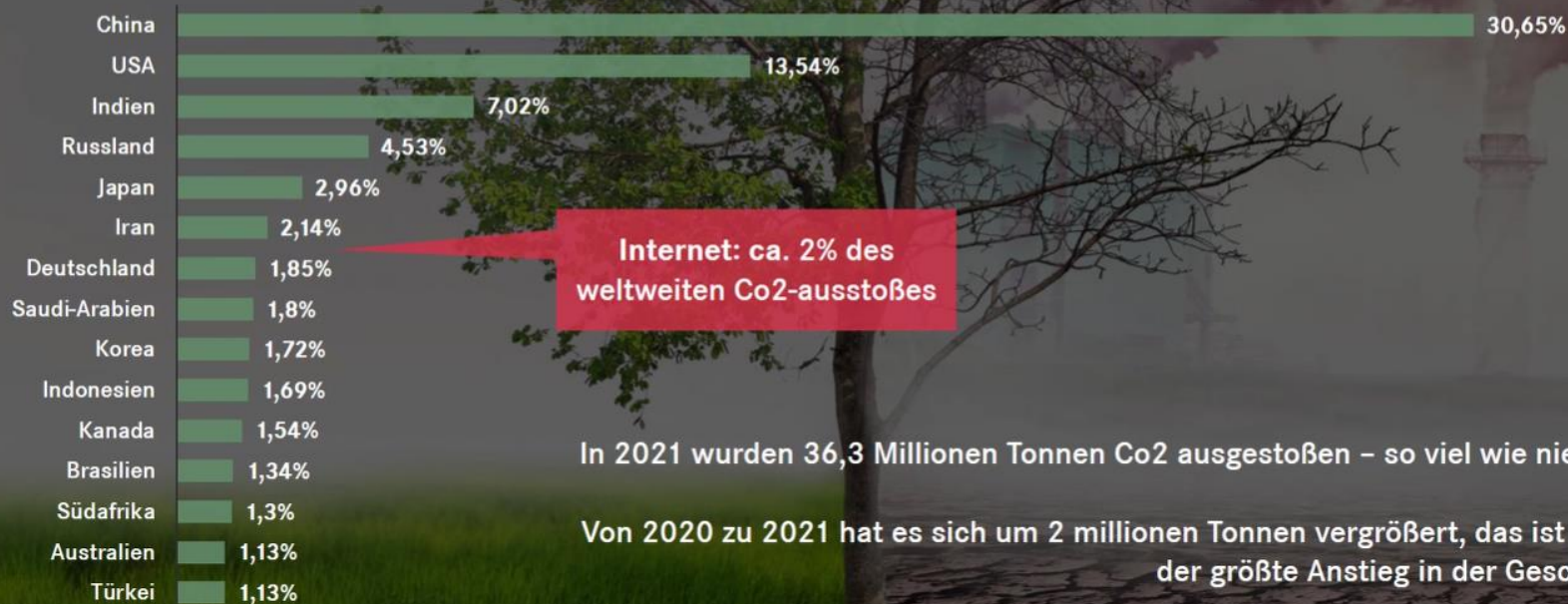


# 1. Block 9:00h - 10:30h

---



Wenn das Internet ein Land wäre, wäre es weltweit der 7. größte Umweltverschmutzer, welches sich bis 2030 drastisch vergrößern soll.



In 2021 wurden 36,3 Millionen Tonnen Co2 ausgestoßen – so viel wie nie zuvor.

Von 2020 zu 2021 hat es sich um 2 Millionen Tonnen vergrößert, das ist mit 6% der größte Anstieg in der Geschichte.

Harvard Business Review,  
September 2020  
*How green is your software* – Es wird geschätzt, dass bis 2030 8% der CO<sub>2</sub>-Emissionen aus dem IT- und Kommunikationssektor entstehen.

# Weltweite Emission – absolute Zahlen [\[Quelle: EU-Kommission\]](#)

Rangliste der weltweiten CO<sub>2</sub>-Emittenten

Ranking	CO <sub>2</sub> -Emissionen in Mio. t	globaler Anteil in %
1. China	11.256	29,7
2. USA	5.275	13,9
3. Indien	2.622	6,9
4. Russland	1.748	4,6
5. Japan	1.199	3,2
6. Deutschland	753	2,0
7. Iran	728	1,9
8. Südkorea	695	1,8
9. Saudi-Arabien	625	1,7
10. Kanada	594	1,6

Quelle: [EU-Kommission](#) : Fossil CO<sub>2</sub> and GHG emissions of all world countries, 2019 report. Bezugsjahr: 2018

Hinweis:  
Ziel für Deutschland gemäß  
Novelle vom 21.05.2021:  
438 Mio. t/Jahr 2030 und  
Treibhausgas-neutral 2045  
[\[1\]](#).



# 1 Suche in Google verursacht soviel CO<sub>2</sub> wie die Produktion von 0.35 gr Rindfleisch [1, 2]



In einer Stunde werden weltweit 228 Mio. Google-Suchen [3] gemacht.

Das entspricht 1.6 Mio. kg CO<sub>2</sub>

Das entspricht der Produktion von 79.800 kg Rindfleisch.

Das entspricht etwa 476 Rindern [4].

-> Wieviel km mit dem Auto sind das? (Mittelklasse, Landstr. ) [5]

# Stromverbrauch durch Telekommunikation & IT [1]

Mehr Energieverbrauch in Haushalten für IT und Kommunikation

Im Vergleich zum Jahr 2000 gab es insbesondere in den Anwendungsfeldern **Informations- und Kommunikationstechnik** sowie Kühl- und Gefriergeräte deutliche Veränderungen. Während sich der Stromverbrauch von Kühl- und Gefriergeräten in den vergangenen zwanzig Jahren fast halbiert hat, **ist der Verbrauch der Informations- und Kommunikationstechnik (IKT) auf mehr als das Doppelte angestiegen.**

Eine Trendwende in nicht in Sicht: Eine Metastudie des französischen Think Tanks „The Shift Project“, die 170 einzelne Untersuchungen zu dem Thema ausgewertet hat, geht von einem jährlichen Anstieg der Energieaufwendungen für **IKT von rund 9 %** aus.

# Der blaue Engel für Software-Produkte (Hochschule Trier & Umweltbundesamt)



Wie misst man die Energie-Effizienz von Software-Produkten ?

Der blaue Engel vergibt in seiner ersten Version Zertifikate für Desktop Software.

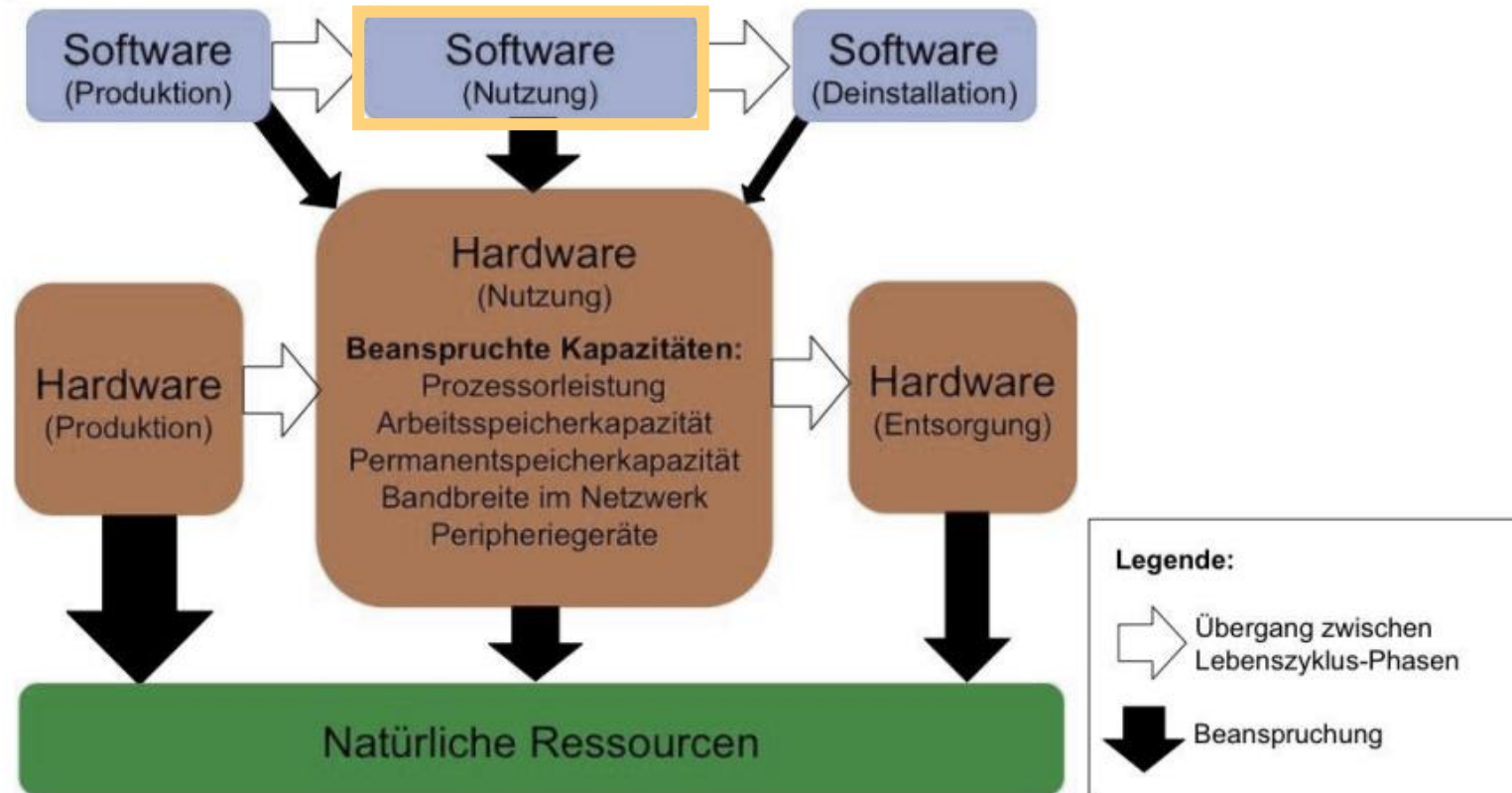
Erstes zertifiziertes Produkt ist der PDF-Viewer Okular von KDE.





# Der blaue Engel für Software-Produkte (Hochschule Trier & Umweltbundesamt)

**Abbildung 2: Lebenszyklen von Hardware und Software (horizontale Dimension) und Beanspruchung von Ressourcen (vertikale Dimension)**

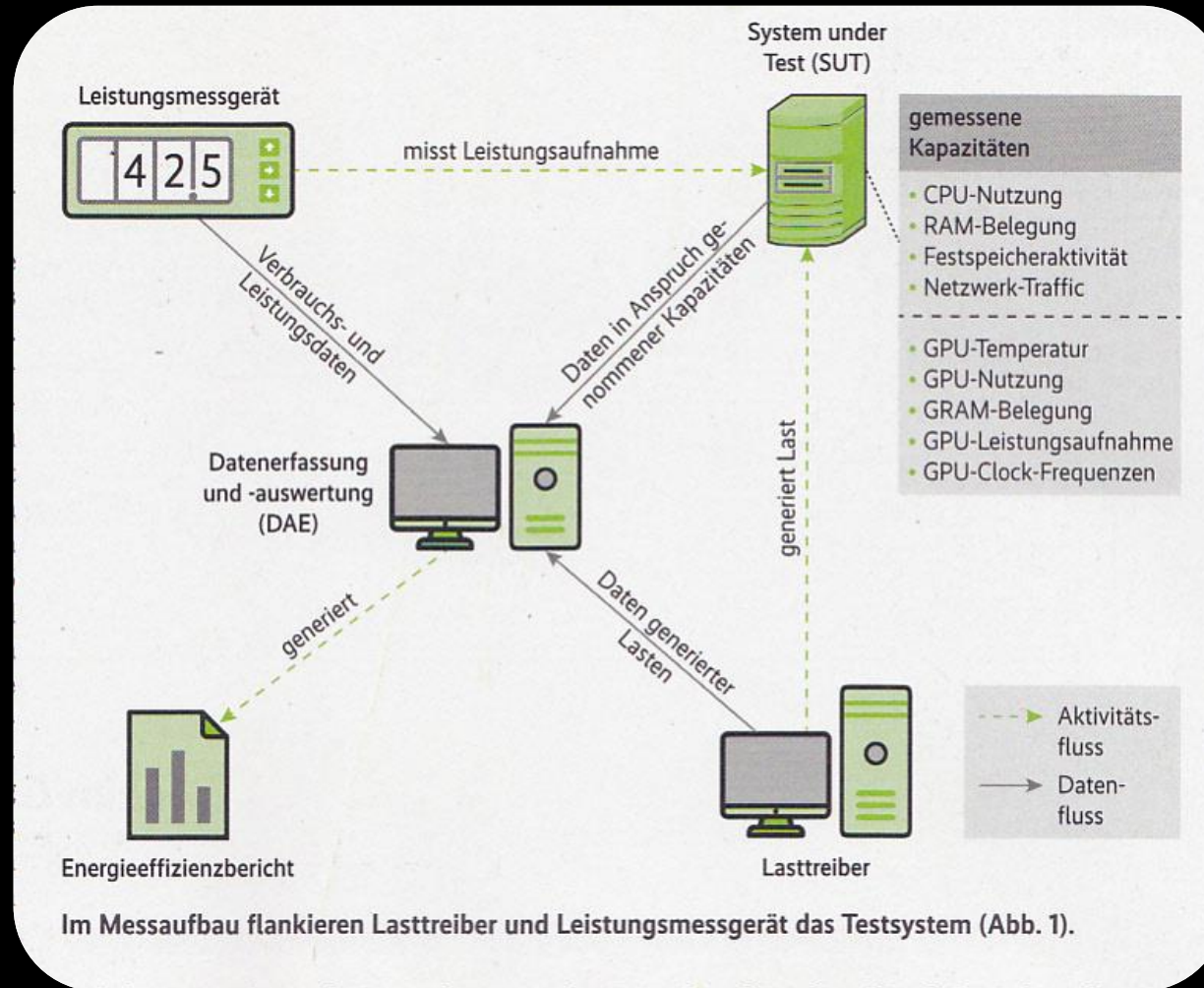


Quelle: Universität Zürich (Gröger et al. 2018)

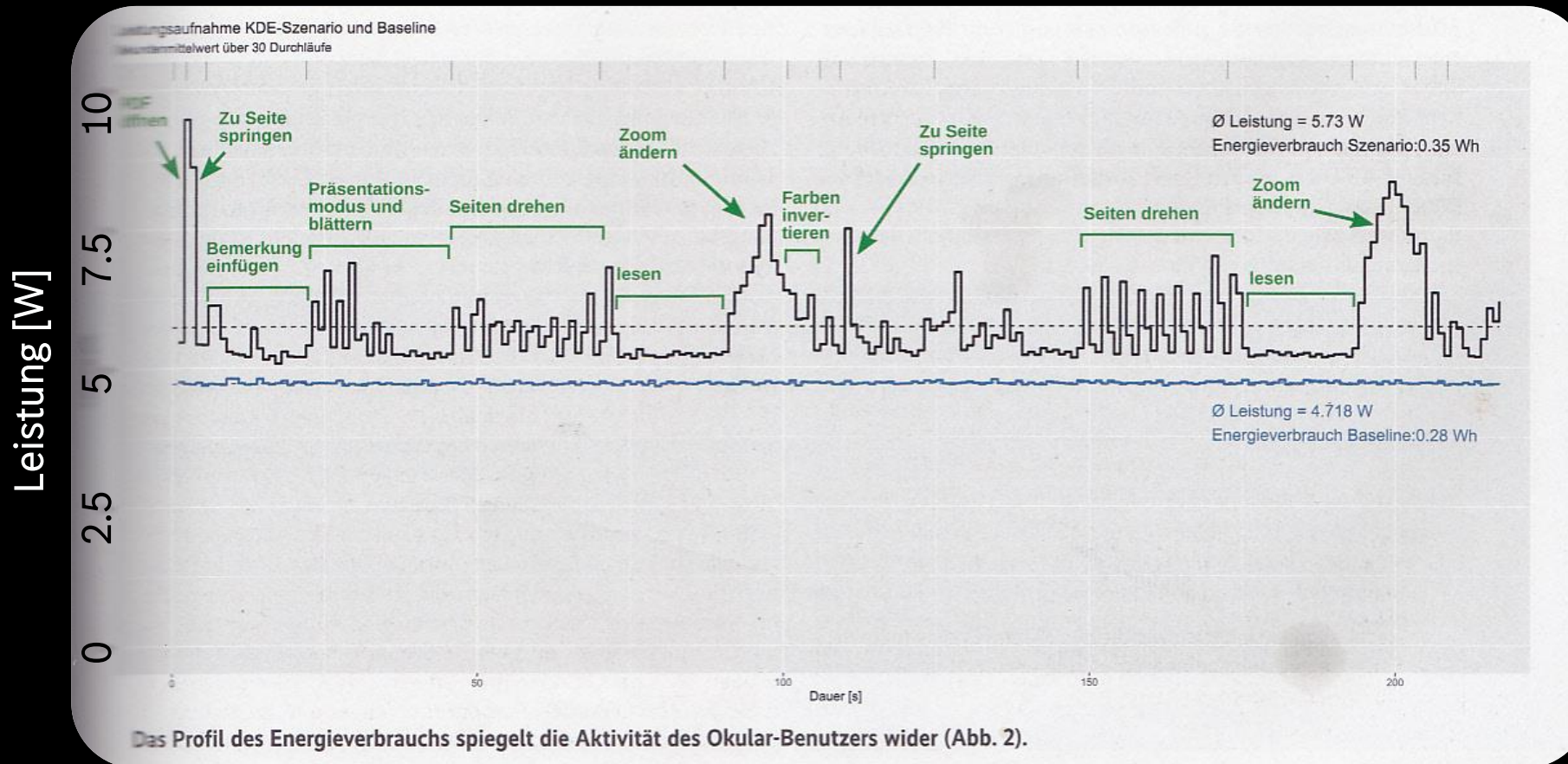


Quelle: Universität Zürich (Gröger et al. 2018)

# Blauer Engel - Messaufbau (entnommen aus iX Spezial 06/2022 Green IT)



# Energieverbrauch für Standard-Szenarien KDE PDF-Viewer „Okular“





## Zitat: Gute Software-Entwicklung ist ...

„Gute Software-Entwicklung für Embedded Systems ist  
einfach gute Software-Entwicklung.“

*(Scott Meyers, aristeria.com: Embedded Systems)*

Das gilt genauso für grüne Softwareentwicklung!

# Prinzipien zur Optimierung von Python Code

## 1. List Comprehension

```
# GPU Intensive code goes here
numbers = []
for x in range(1000000000):
    if x % 2 == 0:
        numbers.append(x**2)
```

## 2. Nutze Built-in Funktionen

```
s = 0
for i in range(1000000000):
    s += 1
```

## 3. Funktionsaufrufe sind teuer ☹️

```
def square(num):
    return num**2

squares = []
for i in range(1000000000):
    squares.append(square(i))
```

# Übung 1: Messung des Energieverbrauches in python mit codecarbon

basic\_testing.py > ...

```
1  from codecarbon import EmissionsTracker
2
3  tracker = EmissionsTracker()
4  tracker.start()
5  # COMPREHENSIVE LISTS
6  # GPU Intensive code goes here
7  numbers = []
8  for x in range(1000000000):
9      if x % 2 == 0:
10         numbers.append(x**2)
11
12  print(' Hallo Informatica femminile 2022')
13
14  tracker.stop()
```

```
18:45:38] Python version: 3.10.5
18:45:38] Available RAM : 63.748 GB
18:45:38] CPU count: 12
18:45:38] CPU model: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz
18:45:38] GPU count: 1
18:45:38] GPU model: 1 x Quadro T1000
Feminale 2022
18:45:39] Energy consumed for RAM : 0.000000 kWh. RAM Power : 23.9
```

```
[codecarbon INFO @ 18:45:39] Energy consumed for all GPUs : 0.000000 kWh. All GPUs Power : 7.416 W
```

```
[codecarbon INFO @ 18:45:39] Energy consumed for all CPUs : 0.000000 kWh. All CPUs Power : 22.5 W
```

```
[codecarbon INFO @ 18:45:39] 0.000000 kWh of electricity used since the begining.
```

```
(codecarbon) PS C:\Users\SGILLNE\informatica> █
```



# Prinzipien zur Optimierung von Python Code

## 1. List Comprehension

```
# GPU Intensive code goes here
numbers = []
for x in range(1000000000):
    if x % 2 == 0:
        numbers.append(x**2)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
    if "a" in x:
        newlist.append(x)

print(newlist)
```

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

## 2. Nutze Built-in Funktionen

```
s = 0
for i in range(1000000000):
    s += 1
```

## 3. Funktionsaufrufe sind teuer ☹️

```
def square(num):
    return num**2

squares = []
for i in range(1000000000):
    squares.append(square(i))
```

Der python interpreter stellt eine Reihe von optimierten Typen und Funktionen zur Verfügung [\[1\]](#). Wenn möglich, nutzt diese!

# Übung 1 + 2

Messe den Energieverbrauch und die Dauer der Ausführung für die drei Optimierungsvarianten jeweils

(a) vor der Optimierung und

(b) nach der Optimierung.

Was ist die Zeitersparnis in % ?

Was ist die Energieeinsparung in % ?

für: [Sabi89129/greencoding: Unterlagen für die Informatica Feminale 2022 in Freiburg \(github.com\)](https://github.com/Sabi89129/greencoding)

erste\_uebung.py

zweite\_uebung.py

# Prinzipien zur Optimierung von Python Code

## 1. List Comprehension

```
# GPU Intensive code goes here
numbers = []
for x in range(1000000000):
    if x % 2 == 0:
        numbers.append(x**2)
```

wird zu

```
numbers = [x**2 for x in range(1000000000) if x % 2 == 0]
```

## 2. Nutze Built-in Funktionen

```
s = 0
for i in range(1000000000):
    s += 1
```

wird zu

```
s = sum(range(1000000000))
```

## 3. Funktionsaufrufe sind teuer ☹️

```
def square(num):
    return num**2

squares = []
for i in range(1000000000):
    squares.append(square(i))
```

wird zu

```
def squares():
    s = []
    for i in range(1000000000):
        s.append(i**2)
    return s

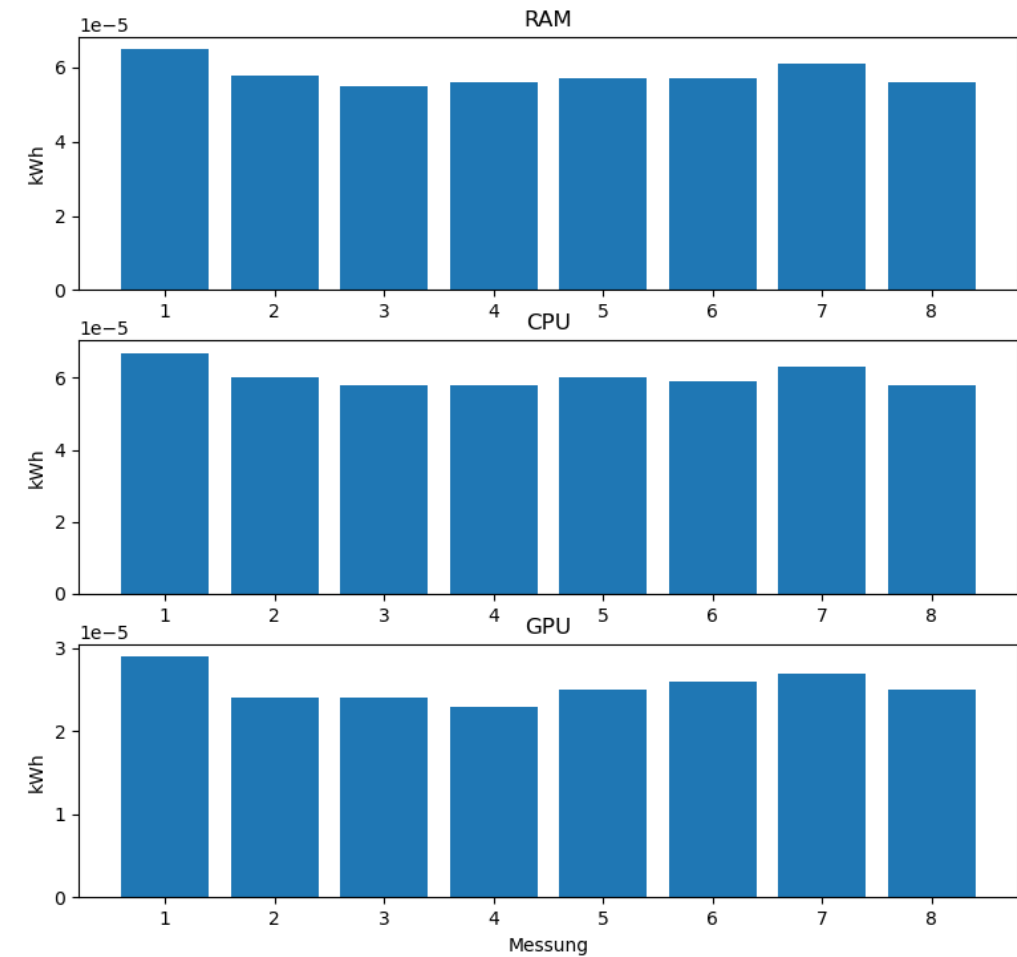
squares()
```



```

firstresult.py > ...
1  import matplotlib.pyplot as plt
2  import numpy as np
3
4  ram = [0.000065,0.000058,0.000055,0.000056,0.000057,0.000057,0.000061,0.000056]
5  GPU = [0.000029,0.000024,0.000024,0.000023,0.000025,0.000026,0.000027,0.000025]
6  CPU = [0.000067,0.000060,0.000058,0.000058,0.000060,0.000059,0.000063,0.000058]
7  print(len(ram))
8  x = np.arange(1,len(ram)+1)
9  print(x)
10
11  plt.subplot(3,1,1)
12  n=len(ram)+2
13  plt.locator_params(axis='x', nbins=n)
14  plt.bar(x,ram)
15  plt.title("RAM")
16  plt.ylabel("kWh")
17
18
19  plt.subplot(3,1,2)
20  plt.locator_params(axis='x', nbins=n)
21  plt.bar(x,CPU)
22  plt.title("CPU")
23  plt.ylabel("kWh")
24
25  plt.subplot(3,1,3)
26  plt.locator_params(axis='x', nbins=n)
27  plt.bar(x,GPU)
28  plt.title("GPU")
29  plt.ylabel("kWh")
30
31  plt.xlabel("Messung")
32
33  |
34  plt.show()

```



## 2. Block 11:00h - 12:30h

---



# Watt & Joule

## Watt

1 Watt ist die **Leistung**, die benötigt wird, um ...

- 1 Gramm Wasser um 14,3 Kelvin zu erwärmen
- pro Sekunde die Arbeit von einem Joule zu verrichten
- $1 \text{ W} = 1 \text{ J/s}$

## Joule

1 Joule ist die **Energie**, die benötigt wird, um ...

- für die Dauer von 1 Sekunde die Leistung von 1 Watt aufzubringen.
- $1 \text{ J} = 1 \text{ N} \cdot \text{m} = 1 \text{ kg} \cdot \text{m}^2/\text{s}^2$

$$\text{Energy (J)} = \text{Power (W)} \times \text{Time(s)}$$

# Wattstunden

## Wattstunde

Die Wattstunde wird alternativ zum Joule genutzt. Sie gehört nicht zum SI-System (internationales Einheitensystem). Im Alltag gebräuchlich ist die Kilowattstunde (kWh).

- $1 \text{ kWh} = 1000 \text{ Wh}$
- $1 \text{ Wattstunde Wh} = 3.6 \text{ kWs} = 3.6 \text{ kJ}$





# Wie berechnet sich nun die Umweltbelastung, die durch 1 kWh entsteht?

» Presse » [Pressemitteilungen](#) » Bilanz 2019: CO<sub>2</sub>-Emissionen pro Kilowattstunde Strom sinken weiter

## Bilanz 2019: CO<sub>2</sub>-Emissionen pro Kilowattstunde Strom sinken weiter Deutschland verkauft mehr Strom ins Ausland als es importiert



Deutschland verkauft weiterhin mehr Strom ins Ausland als es importiert.

Quelle: Gina Sanders / Fotolia.com

Pressemitteilung  
Umweltbundesamt 2019:

Deutschland emittiert 2017  
durchschnittlich **485 gr CO<sub>2</sub>** pro  
kWh.

# Was sind CO<sub>2</sub>-Äquivalente?

CO<sub>2</sub> ist nicht das einzige Treibhausgas, das den Klimawandel antreibt. Es gibt jede Menge anderer Gase, die zur Klimaerwärmung beitragen. Diese werden in den CO<sub>2</sub>-Äquivalenten zusammenfasst. Als Einheit findet man auch CO<sub>2</sub>e.

Man fasst dabei die Treibhausgaswirkung folgender Gase zusammen:

- Kohlendioxid CO<sub>2</sub>
- Methan CH<sub>4</sub>
- Di-Stickstoffoxid N<sub>2</sub>O
- Fluorkohlenwasserstoffe HFC
- Perfluorkohlenwasserstoffe PFC
- Schwefelhexafluorid SF<sub>6</sub>
- Stickstofftrifluorid NF<sub>3</sub>

# Was sind CO<sub>2</sub>-Äquivalente?

Jedes Gas hat dabei ein unterschiedliches Potential zur Erderwärmung beizutragen – das sog. Treibhausgaspotential GWP.

Das GWP gibt das Ausmaß der Erwärmung an, die ein Gas über einen bestimmten Zeitraum verursacht. Für das CO<sub>2</sub>e wird die Wirkung der Gase auf die Erderwärmung auf der Basis des GWP berechnet. Dazu werden die Mengen anderer Gase in die äquivalente Menge von CO<sub>2</sub> umgerechnet.

## Kyoto Gases

(IPCC 2007)

Greenhouse Gas	Global Warming Potential (GWP)
Carbon dioxide (CO <sub>2</sub> )	1
Methane (CH <sub>4</sub> )	25
Nitrous oxide (N <sub>2</sub> O)	298
Hydrofluorcarbons (HFCs)	124 – 14,800
Perfluorocarbons (PFCs)	7,390 – 12,200
Sulfur hexafluoride (SF <sub>6</sub> )	22,800
Nitrogen trifluoride (NF <sub>3</sub> )	17,200

Wenn also 1 kg Methan emittiert wird, kann dies als 25 kg CO<sub>2</sub>e ausgedrückt werden  
(1 kg CH<sub>4</sub> \* 25 = 25 kg CO<sub>2</sub>e).

# Codecarbon: Emissions-Tracker für python

Mit dem Python-Paket codecarbon kann der Verbrauch eines Python-Programms in kWh und CO<sub>2</sub>-Äquivalenten Software-basiert ermittelt werden.

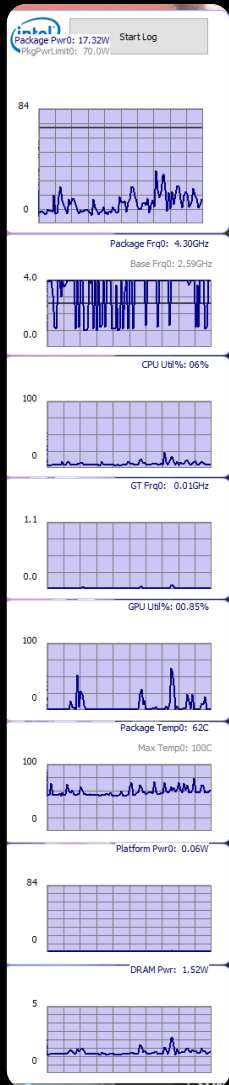


<https://github.com/mlco2/codecarbon>

<https://mlco2.github.io/codecarbon/methodology.html>



# Intel Power Gadget



Software-basierte Messung des Energieverbrauches von CPU, GPU und RAM

Voraussetzung:

<https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html>

# Das paket numPy in python [1]

## PERFORMANT

The core of NumPy is well-optimized C code.

Enjoy the flexibility of Python with the speed of compiled code.

```
import numpy as np

a = np.arange(15).reshape(3, 5)
print('Numpy-Array\n',a)
print('Shape',a.shape)
print('Dimensionen',a.ndim)
print('Typ der Elemente',a.dtype.name)
print('Größe in Byte',a.itemsize)
print('Größe des Arrays',a.size)
print('Typ',type(a))
print('Summe',np.sum(a))
print('Und noch eine Zufallszahl:',np.random.randint(low=0,high=100,))
```

```
Numpy-Array
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
Shape (3, 5)
Dimensionen 2
Typ der Elemente int32
Größe in Byte 4
Größe des Arrays 15
Typ <class 'numpy.ndarray'>
Summe 105
Und noch eine Zufallszahl: 54
```

# Übung 2 + 3

## Summenbildung

```
n = 1000000000
summe = 0
for i in range(n):
    summe = summe+i
```

## Zufallszahlen

```
arr = []
for i in range(n):
    arr.append(random.randint(0,100))
return arr
```

1. Messe jeweils den Energieverbrauch für
  - a. CPU
  - b. GPU
  - c. RAMin mehreren Messungen (n=5)
2. Stelle das Ergebnis graphisch dar, z. B. mit matplotlib
3. Implementiere die for-Schleife mit numPy: Welcher Energieverbrauch zeigt sich jetzt?

# Übung 3 - Lösung

## Summenbildung

```
n = 1000000000
```

```
summe = 0
for i in range(n):
    summe = summe+i
```

## Implementierung mit numPy

```
,np.sum(np.arange(n),dtype=np.float64)
```

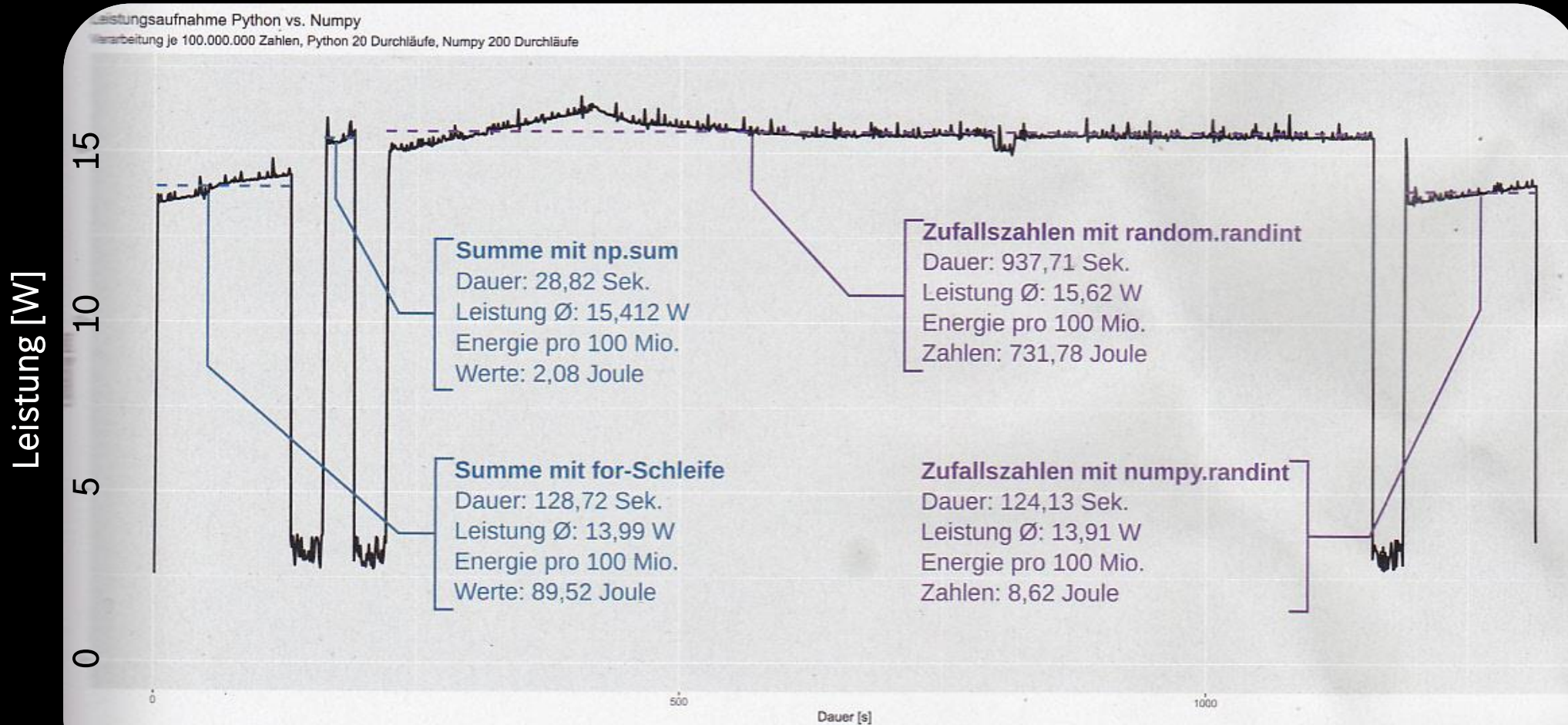
## Zufallszahlen

```
arr = []
for i in range(n):
    arr.append(random.randint(0,100))
return arr
```

```
arr = np.random.randint(low = 0,high = 100, size=(n,))
return arr
```



# Leistungsaufnahme Python vs. NumPy (iX GreenIT 2022, S. 65)

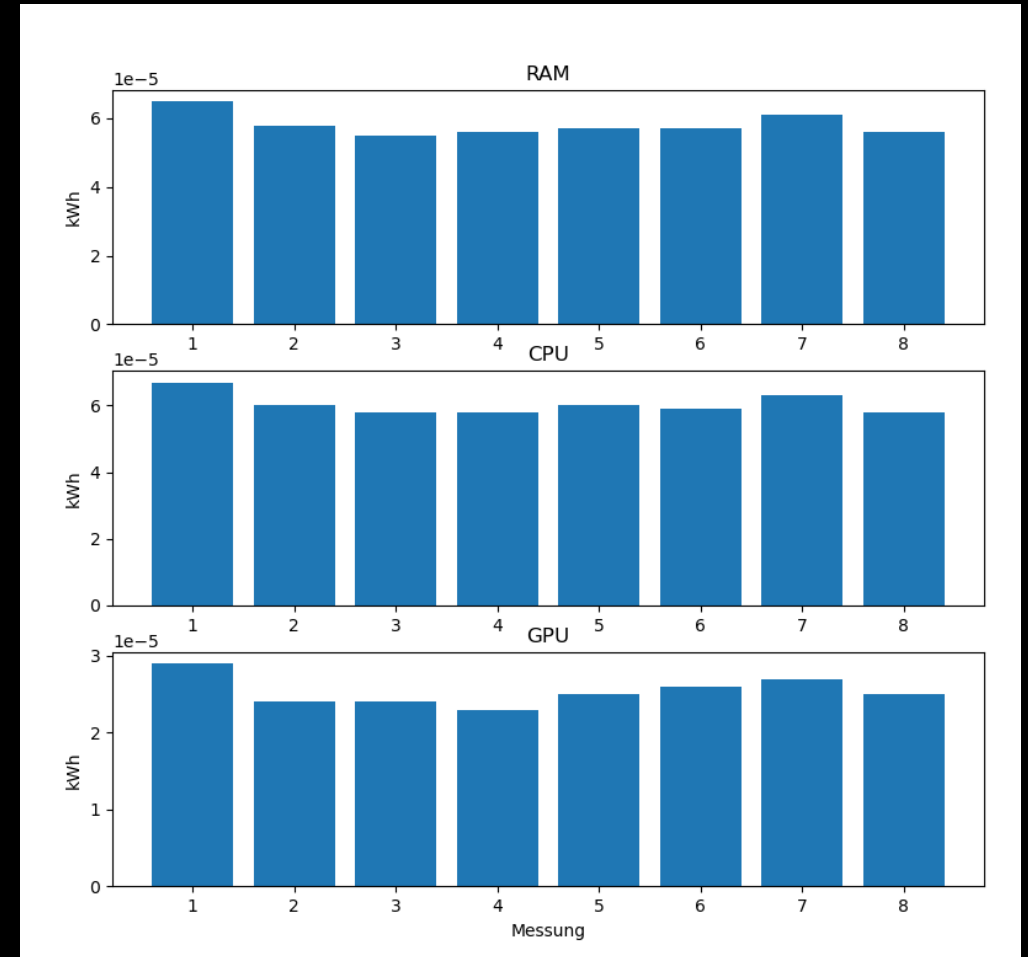


Die Leistungsmessung verrät, wie effizient die Standard-Python- und NumPy-Funktionen arbeiten (Abb. 4).

Die Leistungsmessung verrät, wie effizient die Standard-Python- und NumPy-Funktionen arbeiten (Abb. 4).

# Übung 3 - Zusammenfassung

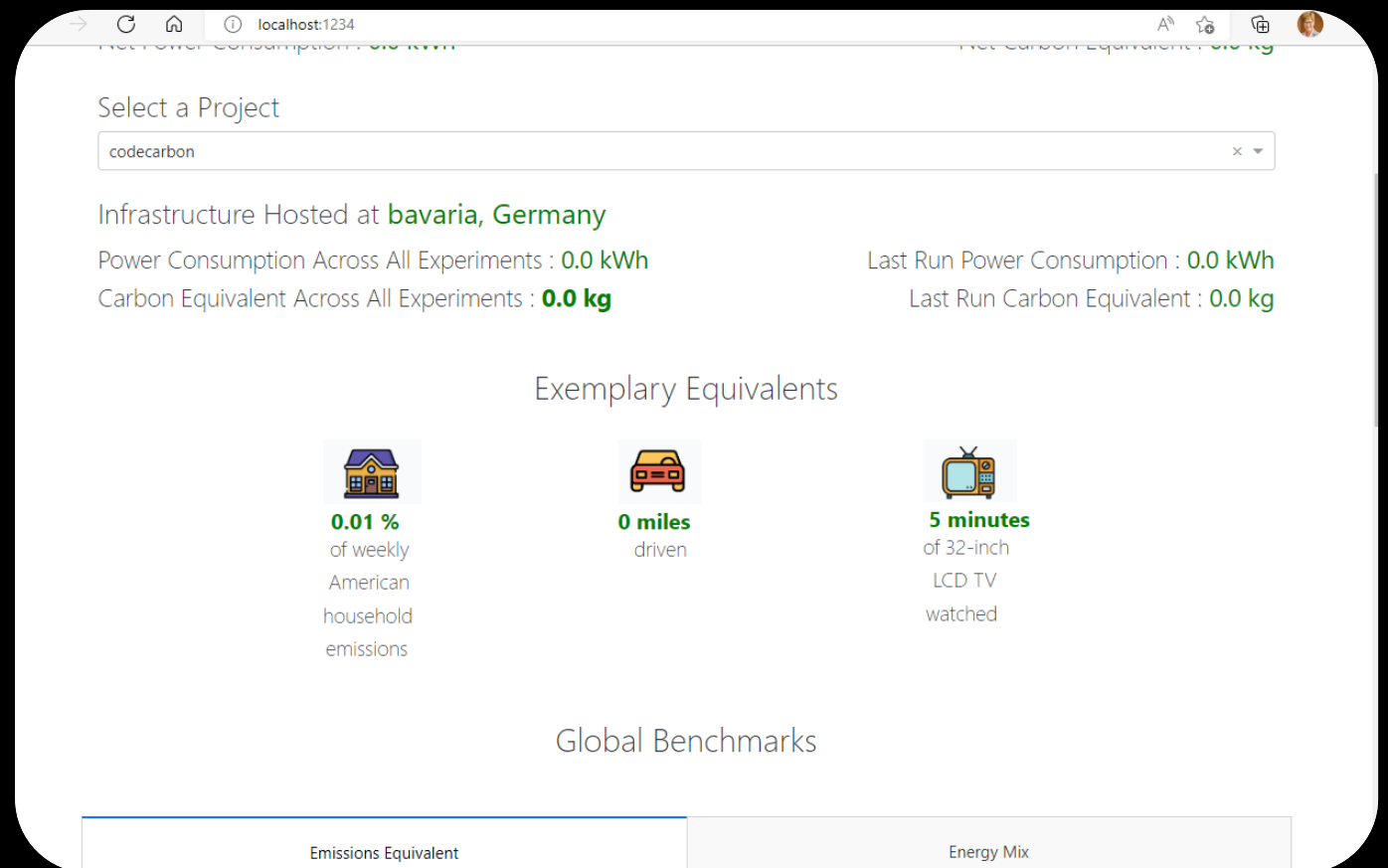
- Mit dem Python-Paket CodeCarbon kann man den Stromverbrauch eines Rechners in kWh messen, getrennt nach RAM, CPU und GPU.
- Der Stromverbrauch variiert bei Programmaufrufen um ca. +/- 5%.
- Mit der Nutzung von NumPy kann zwischen `<insert_your_number_here>` und `<insert_your_number_here>` kWh eingespart werden



# Visualisierung

```
>>> carbonboard --filepath="./emissions.csv" --port=1234
```

Aufruf im Browser mit  
<http://localhost:1234>



### 3. Block 13:30h - 15:00h

---





# Nachhaltige Programmiersprachen ?

---

In einer Studie einer Kooperation verschiedener Universitäten in Portugal wurde die **Energie Effizienz** von 27 Programmiersprachen miteinander verglichen [[2017](#), Update 2020 und 2021]:

*“In fact, in the last century performance in software languages was in almost all cases synonymous of fast execution time (embedded systems were probably the single exception).*

*[....]*

*Nowadays, it is usual to see mobile phone users (which are powerful computers) avoiding using CPU intensive applications just to save battery/energy.”*

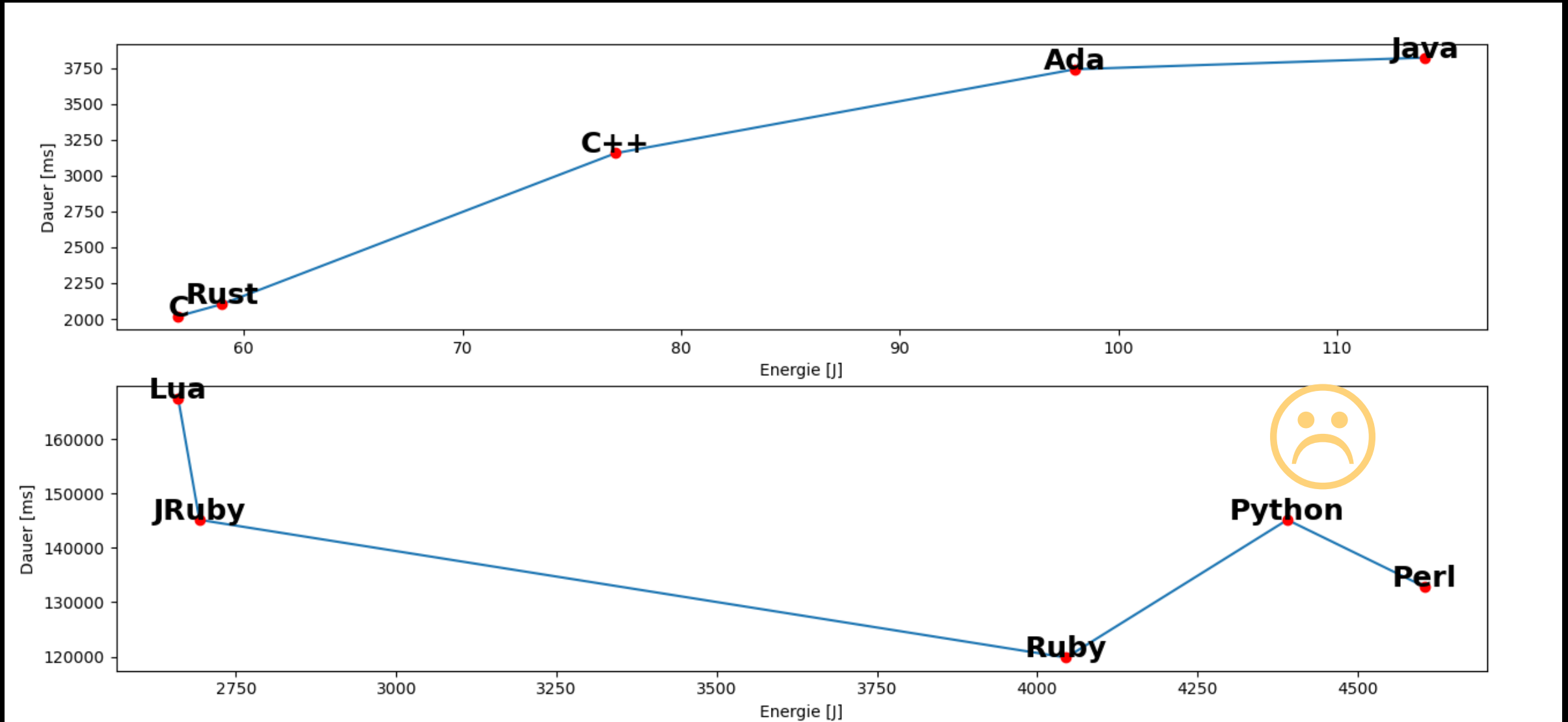
**Table 3.** Results for binary-trees, fannkuch-redux, and fasta

binary-trees				
	Energy	Time	Ratio	Mb
(c) C	39.80	1125	0.035	131
(c) C++	41.23	1129	0.037	132
(c) Rust $\Downarrow_2$	49.07	1263	0.039	180
(c) Fortran $\Uparrow_1$	69.82	2112	0.033	133
(c) Ada $\Downarrow_1$	95.02	2822	0.034	197
(c) Ocaml $\Downarrow_1 \Uparrow_2$	100.74	3525	0.029	148
(v) Java $\Uparrow_1 \Downarrow_{16}$	111.84	3306	0.034	1120
(v) Lisp $\Downarrow_3 \Downarrow_3$	149.55	10570	0.014	373
(v) Racket $\Downarrow_4 \Downarrow_6$	155.81	11261	0.014	467
(i) Hack $\Uparrow_2 \Downarrow_9$	156.71	4497	0.035	502
(v) C# $\Downarrow_1 \Downarrow_1$	189.74	10797	0.018	427
(v) F# $\Downarrow_3 \Downarrow_1$	207.13	15637	0.013	432
(c) Pascal $\Downarrow_3 \Uparrow_5$	214.64	16079	0.013	256
(c) Chapel $\Uparrow_5 \Uparrow_4$	237.29	7265	0.033	335
(v) Erlang $\Uparrow_5 \Uparrow_1$	266.14	7327	0.036	433
(c) Haskell $\Uparrow_2 \Downarrow_2$	270.15	11582	0.023	494
(i) Dart $\Downarrow_1 \Uparrow_1$	290.27	17197	0.017	475
(i) JavaScript $\Downarrow_2 \Downarrow_4$	312.14	21349	0.015	916
(i) TypeScript $\Downarrow_2 \Downarrow_2$	315.10	21686	0.015	915
(c) Go $\Uparrow_3 \Uparrow_{13}$	636.71	16292	0.039	228
(i) Jruby $\Uparrow_2 \Downarrow_3$	720.53	19276	0.037	1671
(i) Ruby $\Uparrow_5$	855.12	26634	0.032	482
(i) PHP $\Uparrow_3$	1,397.51	42316	0.033	786
(i) Python $\Uparrow_{15}$	1,793.46	45003	0.040	275
(i) Lua $\Downarrow_1$	2,452.04	209217	0.012	1961
(i) Perl $\Uparrow_1$	3,542.20	96097	0.037	2148
(c) Swift	n.e.			

fannkuch-redux				
	Energy	Time	Ratio	Mb
(c) C $\Downarrow_2$	215.92	6076	0.036	2
(c) C++ $\Uparrow_1$	219.89	6123	0.036	1
(c) Rust $\Downarrow_{11}$	238.30	6628	0.036	16
(c) Swift $\Downarrow_5$	243.81	6712	0.036	7
(c) Ada $\Downarrow_2$	264.98	7351	0.036	4
(c) Ocaml $\Downarrow_1$	277.27	7895	0.035	3
(c) Chapel $\Uparrow_1 \Downarrow_{18}$	285.39	7853	0.036	53
(v) Lisp $\Downarrow_3 \Downarrow_{15}$	309.02	9154	0.034	43
(v) Java $\Uparrow_1 \Downarrow_{13}$	311.38	8241	0.038	35
(c) Fortran $\Downarrow_1$	316.50	8665	0.037	12
(c) Go $\Uparrow_2 \Uparrow_7$	318.51	8487	0.038	2
(c) Pascal $\Uparrow_{10}$	343.55	9807	0.035	2
(v) F# $\Downarrow_1 \Downarrow_7$	395.03	10950	0.036	34
(v) C# $\Uparrow_1 \Downarrow_5$	399.33	10840	0.037	29
(i) JavaScript $\Downarrow_1 \Downarrow_2$	413.90	33663	0.012	26
(c) Haskell $\Uparrow_1 \Uparrow_8$	433.68	14666	0.030	7
(i) Dart $\Downarrow_7$	487.29	38678	0.013	46
(v) Racket $\Uparrow_3$	1,941.53	43680	0.044	18
(v) Erlang $\Uparrow_3$	4,148.38	101839	0.041	18
(i) Hack $\Downarrow_6$	5,286.77	115490	0.046	119
(i) PHP	5,731.88	125975	0.046	34
(i) TypeScript $\Downarrow_4 \Uparrow_4$	6,898.48	516541	0.013	26
(i) Jruby $\Uparrow_1 \Downarrow_4$	7,819.03	219148	0.036	669
(i) Lua $\Downarrow_3 \Uparrow_{19}$	8,277.87	635023	0.013	2
(i) Perl $\Uparrow_2 \Uparrow_{12}$	11,133.49	249418	0.045	12
(i) Python $\Uparrow_2 \Uparrow_{14}$	12,784.09	279544	0.046	12
(i) Ruby $\Uparrow_2 \Uparrow_{17}$	14,064.98	315583	0.045	8

fasta				
	Energy	Time	Ratio	Mb
(c) Rust $\Downarrow_9$	26.15	931	0.028	16
(c) Fortran $\Downarrow_6$	27.62	1661	0.017	1
(c) C $\Uparrow_1 \Downarrow_1$	27.64	973	0.028	3
(c) C++ $\Uparrow_1 \Downarrow_2$	34.88	1164	0.030	4
(v) Java $\Uparrow_1 \Downarrow_{12}$	35.86	1249	0.029	41
(c) Swift $\Downarrow_9$	37.06	1405	0.026	31
(c) Go $\Downarrow_2$	40.45	1838	0.022	4
(c) Ada $\Downarrow_2 \Uparrow_3$	40.45	2765	0.015	3
(c) Ocaml $\Downarrow_2 \Downarrow_{15}$	40.78	3171	0.013	201
(c) Chapel $\Uparrow_5 \Downarrow_{10}$	40.88	1379	0.030	53
(v) C# $\Uparrow_4 \Downarrow_5$	45.35	1549	0.029	35
(i) Dart $\Downarrow_6$	63.61	4787	0.013	49
(i) JavaScript $\Downarrow_1$	64.84	5098	0.013	30
(c) Pascal $\Downarrow_1 \Uparrow_{13}$	68.63	5478	0.013	0
(i) TypeScript $\Downarrow_2 \Downarrow_{10}$	82.72	6909	0.012	271
(v) F# $\Uparrow_2 \Uparrow_3$	93.11	5360	0.017	27
(v) Racket $\Downarrow_1 \Uparrow_5$	120.90	8255	0.015	21
(c) Haskell $\Uparrow_2 \Downarrow_8$	205.52	5728	0.036	446
(v) Lisp $\Downarrow_2$	231.49	15763	0.015	75
(i) Hack $\Downarrow_3$	237.70	17203	0.014	120
(i) Lua $\Uparrow_{18}$	347.37	24617	0.014	3
(i) PHP $\Downarrow_1 \Uparrow_{13}$	430.73	29508	0.015	14
(v) Erlang $\Uparrow_1 \Uparrow_{12}$	477.81	27852	0.017	18
(i) Ruby $\Downarrow_1 \Uparrow_2$	852.30	61216	0.014	104
(i) JRuby $\Uparrow_1 \Downarrow_2$	912.93	49509	0.018	705
(i) Python $\Downarrow_1 \Uparrow_{18}$	1,061.41	74111	0.014	9
(i) Perl $\Uparrow_1 \Uparrow_8$	2,684.33	61463	0.044	53

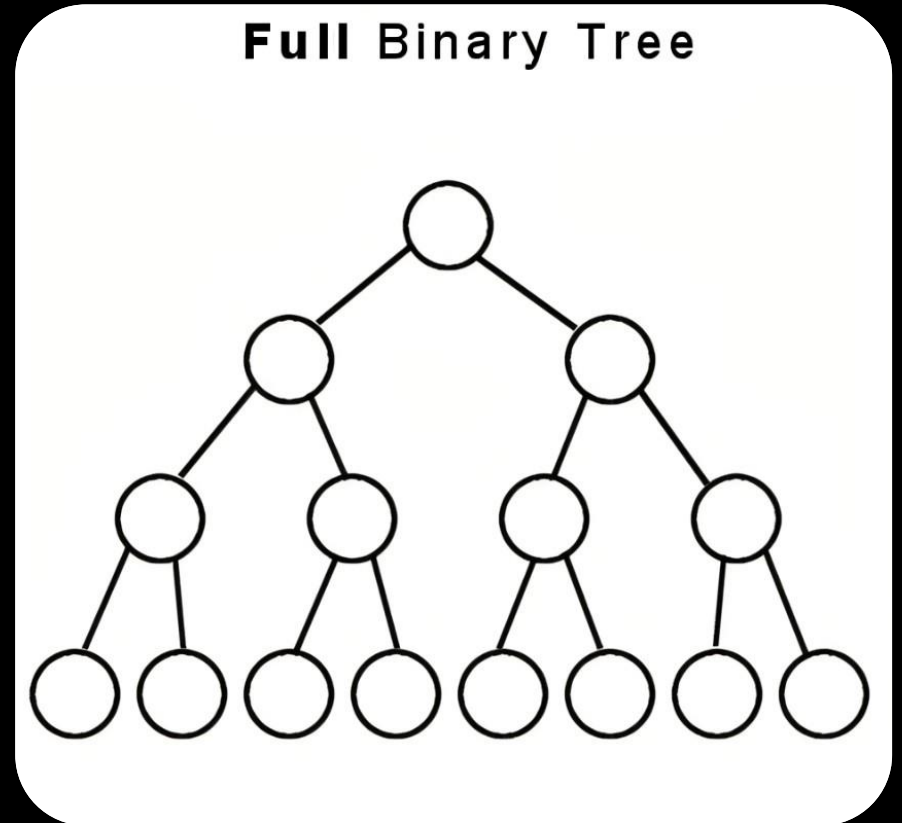
# Nachhaltige Programmiersprachen ?



# Optimiere binarytree

[Energy-Languages/Python/binary-trees at master · greensoftwarelab/Energy-Languages · GitHub](https://github.com/greensoftwarelab/Energy-Languages/tree/master/Python/binary-trees)

1. Messe den Energieverbrauch von `binarytree.py` mit `argv[1] = 21` mit `codecarbon`
2. Vergleiche Dein Ergebnis mit den Zahlen von Peireira et al. [2017](#)



# Kleiner Hackathon

Optimiere vierte\_uebung.py

```
vierte_uebung.py > ...
1  import matplotlib.pyplot as plt
2  import math
3  import random
4
5  # number_of_loops
6  n = 1000000
7
8  # pi
9  pi = 3.14159265359
10
11  x=[]
12  y=[]
13
14  def zahl(i):
15      return i/10000
16
17  def rauschen():
18      sigma = 0.5
19      return random.normalvariate(0,sigma)
20
21  for i in range(n):
22      x.append(zahl(i) * math.cos(zahl(i)) + rauschen())
23      y.append(zahl(i) * math.sin(zahl(i)) + rauschen())
24
25
26  plt.figure(figsize=(15, 15))
27  plt.plot(x,y)
28  plt.show()
29
```



# Let's drive innovation!


Aktuelle  
Stellenangebote:



## Be part of IT!

Mercedes-Benz Tech Innovation





Mercedes-Benz Tech Innovation GmbH  
Wilhelm-Runge-Street 11, 89081 Ulm

Phone +49 731 505-06 / [techinnovation@mercedes-benz.com](mailto:techinnovation@mercedes-benz.com) / [www.mercedes-benz-techinnovation.com](http://www.mercedes-benz-techinnovation.com)

Domicile and Court Registry: Ulm / HRB-No.: 3844 / Management: Daniel Geisel (CEO), Isabelle Krautwald

**Mercedes-Benz Tech Innovation**