



Technische  
Universität  
Braunschweig



# Einführung in die Programmierung für Nicht Informatiker\*innen

Phil Keier  
Institut für Nachrichtentechnik (IfN)  
Technische Universität Braunschweig



# Lösungen 1. Tutorial

31 mögliche Punkte

# Aufgabe Datentypen

Gebe den Text Hallo Python aus, Nutze dabei die print-funktion. 1 Punkt

```
print("Hallo Python")
```



# Aufgabe Datentypen

Aufgabe 2 Punkte:

Definieren Sie zunächst die zwei Variablen a und b und initialisieren diese mit einem Integerwert ungleich 0:

```
a = 1  
b = -2
```

# Aufgabe Datentypen

Aufgabe 2 Punkte:

Definieren Sie zwei Variablen s und t und initialisieren diese mit einem Floatwert ungleich 0:

```
s = 1.5  
t = -2.7
```

# Aufgabe Datentypen

Aufgabe 2 Punkte:

Addieren Sie die Werte der Variablen a und b und speichern Sie das Ergebnis in der Variable c:

```
c = a + b
```

# Aufgabe Datentypen

Aufgabe 5 Punkte:

Nutzen Sie die Variablen  $a$  &  $b$  und Speichern Sie die Ergebnisse für die Multiplikation, Division, Ganzzahldivision, Exponentiation und den Modulo-Operator in den unten angegebenen Variablen:

$$m = a \cdot b$$

$$d = \frac{a}{b}$$

$$i = \left\lfloor \frac{a}{b} \right\rfloor$$

$$e = a^b$$

$$r = a \bmod b$$

$$m = a * b$$

$$d = a / b$$

$$i = a // b$$

$$e = a ** b$$

$$r = a \% b$$

# Aufgabe Datentypen

Aufgabe 2 Punkte:

Ein String-Objekt (Text) können sie mit Hilfe von 'Some Text' oder "Some Text2" definieren. Definieren sie die Variable text mit einem beliebigen Text.

```
text = "Hi Mom, I am on TV!"
```



# Aufgabe Datentypen

Aufgabe 1 Punkte:

Geben Sie die Variablen a & b aus Aufgabe 1 im format "a = 12 und b = 12" (Die Werte sollen dann den Werten aus ihrer Definition entsprechen. 12 ist hier nur ein Beispiel) aus.

```
print("a = {} und b = {}".format(a, b))
```

# Aufgabe Listen

Aufgabe 1 Punkte:

Definieren Sie die Variable `l` und weisen Sie dieser Variable eine Liste mit aufsteigenden Integerwerten von 0 bis 4 zu.

```
l = list(range(5))  
l = [0, 1, 2, 3, 4]
```

# Aufgabe Listen

Aufgabe 1 Punkte:

Hängen Sie der Liste l noch den Wert 42 an.

Hinweis: Nutzen Sie dafür die Methode .append.

```
l.append(42)
```

# Aufgabe Listen

Aufgabe 1 Punkte:

Geben Sie das dritte Element der Liste `l` aus.

Hinweis: Achten Sie darauf das der erste Index immer `0` ist.

```
print(l[2])
```

# Aufgabe Listen

Aufgabe 1 Punkte:

Geben Sie das vorletzte Element der Liste ` ` aus.

Hinweis: Achten Sie darauf das der letzte Index mit -1 ausgegeben wird.

```
print(l[-2])
```

# Aufgabe Dictionaries

Das Dictionary ist ein Datentyp, welcher Schlüssel-Werte-Paare speichert. Dabei wird ein Dictionary mit `dict()` oder `{"Schlüssel1": "Wert1"}` initialisiert. Wichtig ist hierbei das ein Dictionary nicht mit `{}` initialisiert werden kann da dies die Notation für das Set Objekt ist.

Aufgabe 1 Punkte:

Initialisieren Sie die Dictionary Variable ``my_dict`` mit folgendem Mapping:

Key	Value
„apple“	„Apfel“
„banana“	„Banane“
„cherry“	„Kirsche“

```
my_dict = {  
    "apple": "Apfel",  
    "banana": "Banane",  
    "cherry": "Kirsche"  
}
```



# Aufgabe Dictionaries

Aufgabe 1 Punkte:

Fügen Sie nun das Key-Value Paar "pear": "Birne" zu my\_dict hinzu.

```
my_dict["pear"] = "Birne"  
my_dict.update({"pear": "Birne"})
```

# Aufgabe Dictionaries

Aufgabe 2 Punkte:

Speichere die Werte des Dictionaries `my_dict` in der Variablen `values`.

Speichern Sie die Elemente des Dictionaries `my_dict`, welche mit der Funktion `.items()` ausgegeben werden, in der Variablen `items`.

```
values = my_dict.values()  
items = my_dict.items()
```

# Aufgabe Funktionen

Aufgabe 1 Punkte\*:

Schreibe eine Funktion successor die auf jede Eingabe +1 rechnet.

Schreibe eine Funktion add mit den Eingabeparametern a & b, welche die Werte von a & b miteinander addiert.

```
def successor(n):  
    return n+1  
  
def add(a,b):  
    return a+b
```

# Aufgabe Funktionen

Aufgabe 1 Punkte:

Schreibe eine Funktion `is_odd` mit einem Eingabeparameter `n` die prüft ob die eingegebene Zahl ungerade ist.

Wenn die Zahl gerade ist gebe den Text "Gerade Zahl" und bei ungerade "Ungerade Zahl" zurück.

```
def is_odd(n):  
    if n % 2 == 0:  
        return "Gerade Zahl"  
    else:  
        return "Ungerade Zahl"
```

# Aufgabe Funktionen

7 Punkte.

Aufgabe: Schreibe eine Funktion fubar mit Eingabeparameter n.  
Die Funktion soll wie folgt definiert sein:

- Der Eingabeparameter n ist ein Integer, Floats geben False zurück
- Negative zahlen & 0 beenden die Funktion und geben False zurück
- Die Funktion zählt bis einschließlich dem Eingabeparameter n = 9 -> 1, 2, 3, ..., 9
- Bei jedem Schleifendurchlauf soll die Zahl bei der sich die Schleife gerade befindet mittels print ausgegeben werden werden.
- Ist der zurzeitige Schleifendurchlauf durch 3 teilbar, gebe mittels print denn String Foo aus.
- Ist der zurzeitige Schleifendurchlauf durch 5 teilbar, gebe mittels print denn String Bar aus.
- Ist der zurzeitige Schleifendurchlauf durch 3 & 5 teilbar, gebe mittels print den String FooBar aus.

# Aufgabe Funktionen

```
def fubar(n: int):  
    if isinstance(n, float) or n < 1:  
        return False  
  
    count = 1  
    while count <= n:  
        msg = count  
        if count % 3 == 0:  
            msg = "Foo"  
        if count % 5 == 0:  
            msg = "Bar"  
        if count % 15 == 0:  
            msg = "FooBar"  
  
        count += 1  
        print(msg, end=', ')
```





# Lösungen 2. Tutorial

17+1 mögliche Punkte

# Aufgabe - Kontrollstrukturen

3 Punkte

Schreibe eine Funktion `sum_up` mit Eingabeparameter `n`, welcher die Zahlen von `1...n` aufsummiert.

Nutze dafür einen `for-loop`.

```
def sum_up(n: int) -> int:
    count = 0
    for i in range(1, n+1):
        count += i
    return count
```

# Aufgabe - Kontrollstrukturen

2 Punkte

Gegeben ist das Dictionary dict2.

Ändere jeden Wert in diesem Dictionary entsprechend der Formel  $f(x) = x^3 - 1$  mittels for-loop.

```
dict2 = {"a": 56, 5: 12, "python": 9, 3.14: 1.141414}

# Möglichkeit 1 - Dictionary Comprehension
dict2 = {key: value**3-1 for key, value in dict2.items()}

# Möglichkeit 2 - For Loop
for key, value in dict2.items():
    dict2[key] = value**3 - 1
```

# Zusatzaufgabe - Kontrollstrukturen

Keine Punkte (Extra Punkt möglich)

Erstelle eine Liste mittels List Comprehension, welche die Zahlen 1...6 auf deren kubische Zahl 1...216 also der Funktion  $f(x) = x^3$  abbildet.

```
cubics = [n**3 for n in range(1,7)]
```



# Aufgabe – System Interactions

2 Punkte

Erstelle und Öffne eine Datei testfile.txt mit der open Funktion, nutze dafür das with - Statement.

Schreibe in diese Datei 100 mal den String "Python\n".

```
# Möglichkeit 1
with open('testfile.txt', 'w') as f:
    for _ in range(100):
        f.write("Python\n")
```

```
# Möglichkeit 2
with open('testfile.txt', 'w') as f:
    f.write("Python\n" * 100)
```

# Aufgabe – System Interactions

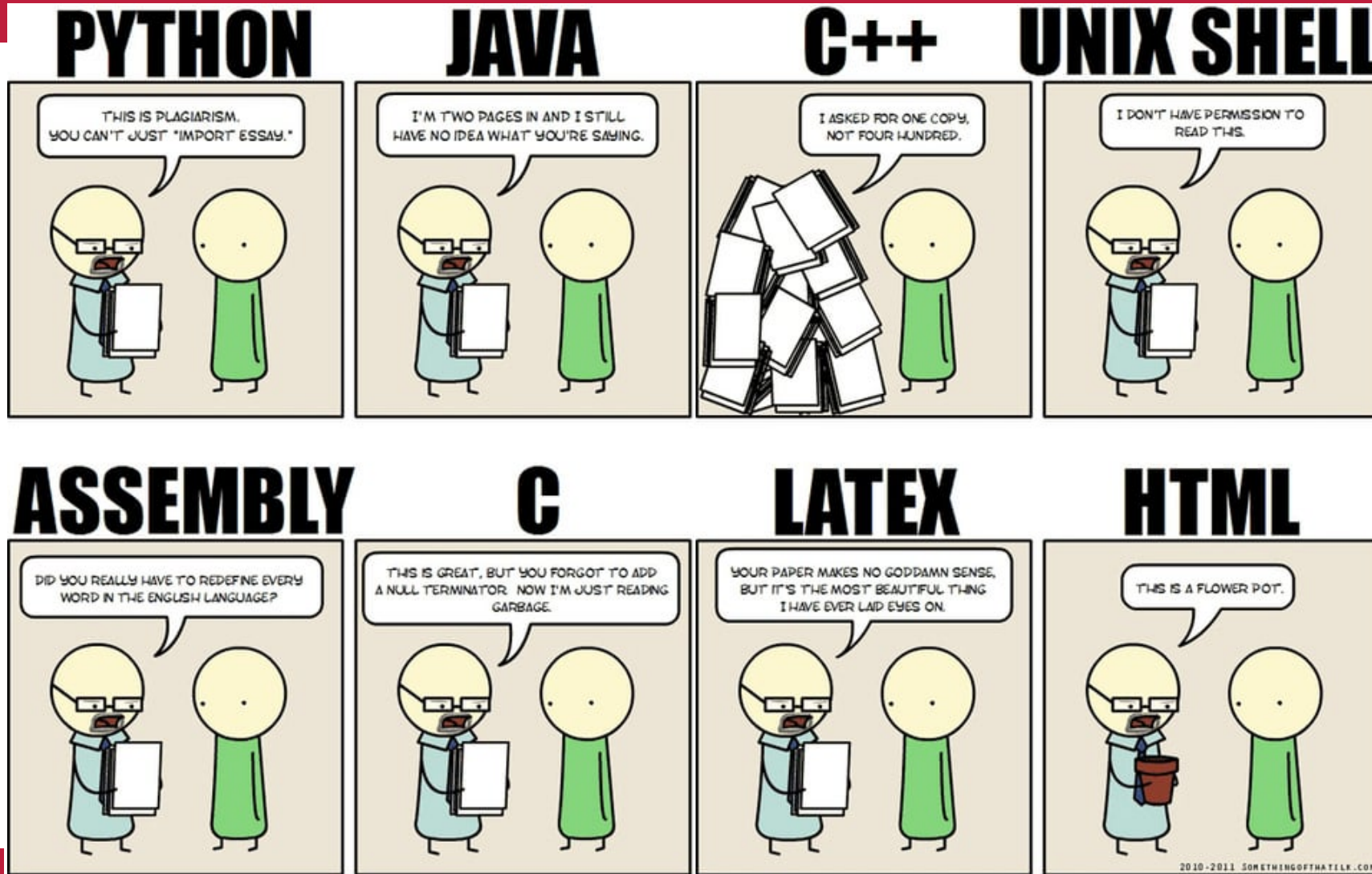
2 Punkte

Öffne die zuvor erstellte Datei testfile.txt im Lesemodus und weiße den Inhalt der .readlines() Funktion der Variablen lines zu.

```
with open('testfile.txt', 'r') as f:  
    lines = f.readlines()
```



# Memes



# Aufgabe - Importing

3 Punkte

Importiere Pythons Built-In Library random und rufe zuerst aus dem Modul die Funktion seed auf mit dem Eingabewert 42, und weiße danach der Variable rand den Wert des Funktionsaufrufes von randint(1,100) zu.

```
import random
random.seed(42)
rand = random.randint(1,100)
```

# Aufgabe - Importing

1 Punkt

Importiere die Built-In Library datetime als dt.

```
import datetime as dt
```

# Aufgabe - Importing

2 Punkte

Importiere die Funktion `sqrt` aus dem Built-In Modul `math`.  
Berechne `sqrt(4)`. Speicher das Ergebnis in der variablen `s4`.

```
from math import sqrt  
s4 = sqrt(4)
```

# Aufgabe - Importing

2 Punkte

Importiere das Objekt TextCalendar aus dem Built-In Module calendar als TC.

```
from calendar import TextCalendar as TC

# Output
    November 2025
Mo Tu We Th Fr Sa Su
|  |  |  |  | 1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 [21] 22 23
24 25 26 27 28 29 30

21.11.2025 - Nächste Vorlesung
```





# Lösungen 3. Extended Applications

26 mögliche Punkte



# Aufgabe Generatoren

3 Punkte

Schreibe einen Generator *square\_count* mit dem Eingabeparameter *n*, der die Quadratzahlen von 1 ...  $(n-1)^2$  erzeugt.

Wichtig: Wenn *square\_count* kein Generator ist, gibt es 0 Punkte.

Hinweis: Bei einer Eingabe von 5 sollen die Werte 1 4 9 16 ausgegeben werden.

# Aufgabe Generatoren

```
def square_count(n):  
    for i in range(1, n):  
        yield i*i  
  
# Test  
for n in range(2, 7):  
    print(f"Square Numbers from 0 to {n-1}:", *square_count(n))
```

## Ausgabe:

Square Numbers from 0 to 1: 1

Square Numbers from 0 to 2: 1 4

Square Numbers from 0 to 3: 1 4 9

Square Numbers from 0 to 4: 1 4 9 16

Square Numbers from 0 to 5: 1 4 9 16 25

# Aufgabe Generatoren

3 Punkte

Schreibe einen Generator *naturals*, der bei jedem Aufruf die nächste natürliche Zahl ausgibt, beginnend mit 1.

- Es sind **keine Eingabeparameter** erforderlich.
- Wenn die Funktion **kein Generator** ist, gibt es **0 Punkte**.

Hinweis: Orientiere dich am Beispiel *faktoriell\_gen*.

```
def faktoriell_gen():  
    curr = 1  
    count = 1  
    while 1:  
        curr = count * curr  
        count += 1  
        yield curr
```

# Aufgabe Generatoren

```
# Test if generator works as intended
import random
test_n = random.randint(5, 17)
test_gen = naturals()
for i in range(1, test_n):
    number = next(test_gen)
    print(number, end=', ')
    assert i == number
```

**Ausgabe** (test\_n = 14):

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,

```
def naturals():
    curr = 1
    while 1:
        yield curr
        curr += 1
```

# Aufgabe Datenklassen

8 Punkte

Erstelle eine Dataclass *Student*.

- Die Hinweise sind aktuell nicht in logischer Reihenfolge.  
Strukturier den Code sinnvoll!

**Beispielstudent:**

Attribut	Wert
vorname	Martin
nachname	Le
mat_nr	520420
semester	5

# Aufgabe Datenklassen

1. Importiere aus dem `dataclasses`-Modul die Funktion `asdict`.
2. Die Dataclass soll die Werte *vorname*, *nachname*, *semester* und *mat\_nr* speichern. Wählen Sie für jedes Attribut den geeigneten Datentyp (z. B. macht es Sinn, die Semesteranzahl als *int* zu speichern, nicht als *float*).
  - **Wichtig:** Wenn *Student* keine Dataclass ist, gibt es **0 Punkte**.
  - Alle Variablen sollen mit *Type Hints* versehen werden.
- Hat eines der Attribute keinen optimalen Datentyp, gibt es trotzdem Punkte, solange die Mehrheit korrekt ist.
3. Erstelle ein Objekt mit den Werten des Beispielstudenten und weise das Ergebnis von `asdict(<beispielstudent>)` der Variablen *stud* zu.
4. Füge Kommentare ein, die sich auf die jeweilige [PEP 8](#) Regel beziehen.

# Aufgabe Datenklassen

```
# Imports on top of the File
from dataclasses import asdict

# Newlines between logic blocks

# Docstring classes to improve understandability
# Classes are written in Title Case
@dataclass
class Student:
    '''
    Dataclass to store student Informations
    '''
    vorname: str
    nachname: str
    mat_nr: int
    semester: int
```

```
# Variables should be Type Hinted,
# if not obvious
stud: dict = asdict(
    # Proper spacing for readability
    Student(
        vorname='Martin',
        nachname='Le',
        mat_nr=520420,
        semester=5
    )
)
```

# Aufgabe Datenklassen

6 Punkte

Gegeben sind zwei Listen *colorn* und *colorv\_hex*, die zueinander indexsortiert sind.

1. Erstelle eine Dataclass *Color* mit den Attributen *name* und *value* und versehe sie mit passenden Type Hints.
2. Erzeuge anschließend eine Liste, die die Werte aus *colorn* und *colorv\_hex* in Instanzen der Dataclass *Color* umwandelt, und speichere diese Liste in der Variablen *colors*.

```
colorn = ['RED', 'GREEN', 'BLUE', 'YELLOW', 'PURPLE']  
colorv_hex = ['#FF0000', '#00FF00', '#0000FF', '#FFFF00', '#FF00FF']
```



# Aufgabe Datenklassen

```
@dataclass
class Color:
    name: str
    value: str
```

Möglichkeit 2:

```
colors: list = []

for i in range(len(color_hex)):
    current_color = Color(
        name=colorn[i],
        value=color_hex[i]
    )
    colors.append(current_color)
```

Möglichkeit 1:

```
colors: list = [Color(n, w) for n, w in zip(colorn, colorv_hex)]
```

# Aufgabe Datenklassen

6 Punkte

Gegeben ist ein Dictionary *dict\_obj*.

1. Erstelle eine Dataclass *Transaction*, die die Originalstruktur des Dictionaries abbildet.

Achte dabei auf **korrekte Type Hint** für die einzelnen Attribute.

2. Schreibe **unterhalb der Dataclass in einer Markdown-Zeile** deine Begründung, warum du diese Struktur gewählt hast. (**!! Nicht als Kommentar !!**)

```
dict_obj: dict = {  
    "IBAN": "DE19500211001598716891", # FAKE!  
    "Transaction Number": random.randint(1, 8000),  
    "Recipient": {  
        "First Name": "Phil",  
        "Last Name": "Keier",  
    },  
    "Amount": 3.14,  
}
```

# Aufgabe Datenklassen

```
@dataclass
class Recipient:
    first_name: str
    last_name: str

@dataclass
class Transaction:
    iban: str
    trans_nr: int
    recipient: Recipient
    amount: float
```

*„Die vorliegende Struktur bildet die Grundlage für die Verarbeitung finanzieller Transaktionen.*

*Für die Repräsentation, aufgeteilt auf zwei separate, jedoch logisch miteinander verknüpfte Datenklassen, wurde bewusst entschieden, um eine hohe Modularität und Wiederverwendbarkeit der Daten sicherzustellen.*

*Das gewählte Modell ermöglicht es, unabhängige Analysen der Empfänger (hier: Recipient) effizient und ohne direkte Abhängigkeit von transaktionsspezifischen Daten durchzuführen, wodurch eine klare Trennung der Verantwortlichkeiten sowie eine verbesserte Wartbarkeit des Systems erreicht wird.“*





# KI-Regeln

# KI Regeln - Programmierung

1. Die Verantwortung für KI-generierten Code obliegt dem/der Studierenden
2. Python Objekte (Funktionen/Klassen/Logische Blöcke) sind mittels *Docstring* kommentiert. Hierbei gilt:
  - Angabe des verwendeten KI Modells inklusive Zugriffsdatum
  - Angabe des verwendeten Prompts
  - Zweck der Nutzung ist zu erläutern
  - Zitations- sowie Verständniskommentare der KI sind zu entfernen
  - Die Einhaltung von PEP8 ist zu überprüfen
3. KI Generierter Code unterliegt gesonderten Styling. **Alle** logischen Blöcke sind mit einem Kommentar zu versehen, der den jeweiligen Zweck kurz beschreibt.

# KI Regeln - Programmierung

Schreibe mir einen Generator, welcher mir die Fakultät einer Zahl berechnet. Nutze dafür Python.

Hier ist ein Python-Generator, der die Fakultät einer Zahl berechnet und dabei die Zwischenergebnisse schrittweise liefert:

```
python Copy code

def fakultaet_generator(n):
    """Generator, der die Fakultät von n Schritt für Schritt berechnet."""
    if n < 0:
        raise ValueError("Die Fakultät ist nur für nicht-negative Zahlen definiert.")

    ergebnis = 1
    for i in range(1, n + 1):
        ergebnis *= i
    yield ergebnis
```

Beispielverwendung:

```
python Copy code

for wert in fakultaet_generator(5):
    print(wert)
```

Ausgabe:

```
Copy code

1
2
6
24
120
```

👉 Der Generator gibt bei jedem Schritt den aktuellen Fakultätswert zurück, bis `n!` erreicht ist.

Wenn du stattdessen nur den finalen Wert (ohne Zwischenschritte) möchtest, sag gerne Bescheid!

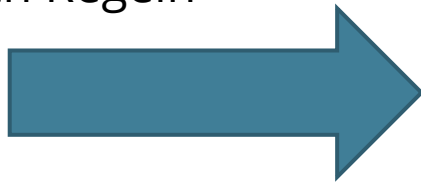


# KI Regeln - Programmierung

```
Python
def fakultaet_generator(n):
    """Generator, der die Fakultät von n Schritt für Schritt berechnet."""
    if n < 0:
        raise ValueError("Die Fakultät ist nur für nicht-negative Zahlen definiert.")

    ergebnis = 1
    for i in range(1, n + 1):
        ergebnis *= i
    yield ergebnis
```

Transformiert nach Regeln



```
Python
def fakultaet_generator(n: int) -> int:
    """
    Generiert mittels ChatGPT (26.11.2025)

    Prompt: "...|"

    - Berechnung der Fakultät
    - Robuste Fehlerbehandlung
    - Benötigt für Berechnung XXX
    """

    # Robuste Fehlerbehandlung bei Werten kleiner 0
    if n < 0:
        raise ValueError("Die Fakultät ist nur für nicht-negative Zahlen definiert.")

    # Berechnung der Fakultät mittels aufmultiplizieren
    ergebnis = 1
    for i in range(1, n + 1):
        ergebnis *= i
    yield ergebnis
```

# KI Regeln - Texte

1. Die Verantwortung für KI-generierten Text obliegt dem/der Studierenden
2. KI generierter/bearbeiteter Text muss gekennzeichnet werden (Datum & Model).
3. Falls KI nur zur Korrektur verwendet wird muss am Ende des Textes der Satz stehen:  
*„Korrektur der Sätze erfolgte mit <KI-Model> am <Datum>“*

Bsp.:

(ChatGPT am 26.11.2025)

*„Diese Funktion kann verwendet werden, um die Fakultät einer Zahl schrittweise zu berechnen und dabei jeden Zwischenwert zu erhalten. Sie ist besonders nützlich, wenn große Fakultäten berechnet werden sollen und man den Fortschritt beobachten oder Zwischenergebnisse weiterverarbeiten möchte. Außerdem eignet sie sich gut für speichereffiziente Berechnungen, da sie die Werte nacheinander generiert, anstatt alles auf einmal zu speichern.“*





# Lösungen 4. NumPy & Matplotlib

45 mögliche Punkte

# Aufgabe Arrays

1 Punkt

Erstelle ein NumPy Array, welches 600 Nullen reserviert und speicher das Array in der Variablen `only_zeros`.

```
only_zeros = np.zeros(600)
```

# Aufgabe Arrays

1 Punkt

Erstelle ein NumPy Array mit 11 Elementen mithilfe der Funktion linspace. Der Startwert soll -6 und der Endwert 16.5 betragen.

Speichere das Ergebnis in der Variablen x\_scale.

```
x_scale = np.linspace(-6, 16.5, num=11)
```

# Aufgabe Arrays

6 Punkt

Erstelle ein NumPy Array und bearbeite es anschließend wie folgt:

1. Erzeuge ein eindimensionales NumPy Array numbers mit 20 gleichmäßig verteilten Zahlen zwischen 10 und 50 mithilfe von np.linspace.
6. Gebe als Kommentar die NumPy Referenz zu den verwendeten Funktionen an. (Keine Kommentare = 0 Punkte)

<https://numpy.org/doc/stable/reference/generated/numpy.linspace.html#numpy-linspace>

```
numbers = np.linspace(10, 50, 20)
```

# Aufgabe Arrays

2. Berechne ein neues Array `squared_numbers`, das die Quadrate der Werte aus `numbers` enthält.

<https://numpy.org/doc/stable/reference/generated/numpy.square.html>

```
squared_numbers = numbers**2  
squared_numbers = np.square(numbers)
```

# Aufgabe Arrays

3. Finde alle Elemente in numbers, die größer als 30 sind, und speichere sie in einem neuen Array numbers\_gt\_30.

<https://numpy.org/doc/stable/user/basics.indexing.html#boolean-indexing>

```
numbers_gt_30 = numbers[numbers > 30]
```



# Aufgabe Arrays

4. Berechne die Summe aller Werte in `squared_numbers` und speichere das Ergebnis in `total_squared`.

<https://numpy.org/doc/stable/reference/generated/numpy.sum.html>

```
total_squared = np.sum(squared_numbers)
```



# Aufgabe Arrays

5. Sortiere numbers absteigend und speichere das Ergebnis in numbers\_sorted\_desc.

<https://numpy.org/doc/stable/reference/generated/numpy.sort.html#numpy.sort>

```
numbers_sorted_desc = np.sort(numbers)[::-1]
```

# Aufgabe Plotting - Squareroot

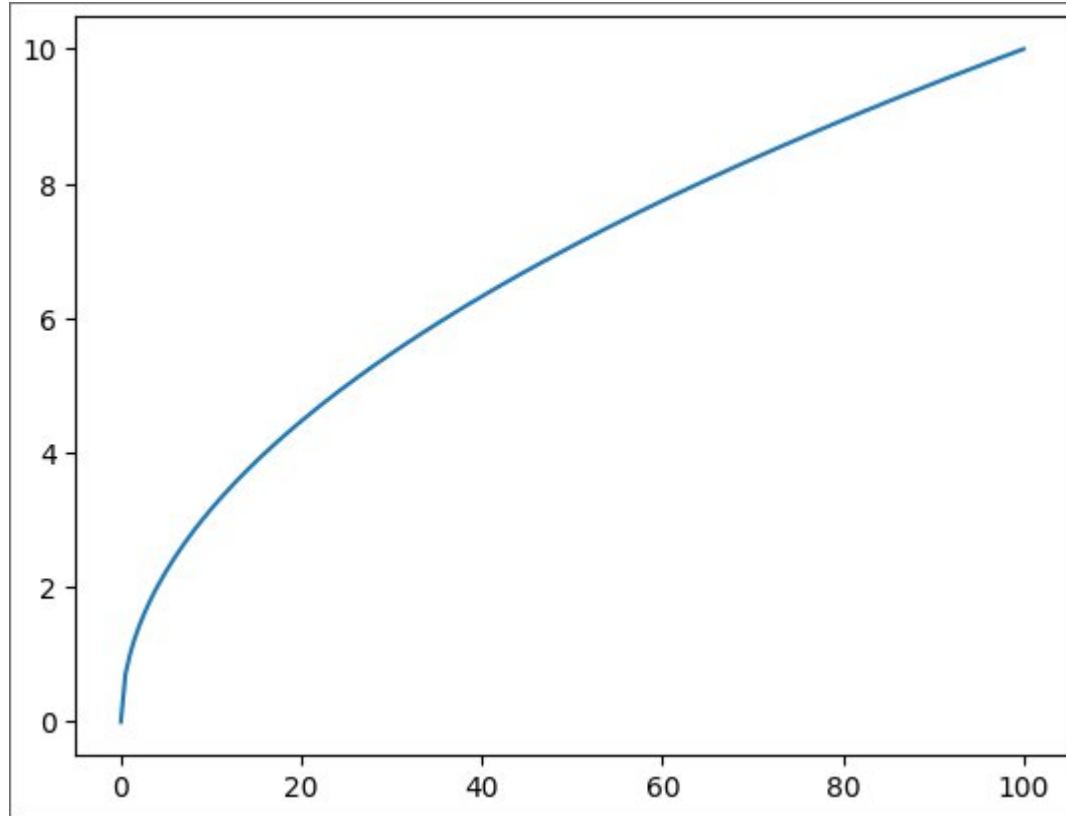
\*3 Punkte\*

Plote im Folgenden die Funktion Square Root, mathematisch definiert als  $f(x) = \sqrt{x}$ ,  $x \geq 0$ .

Gehe dabei wie folgt vor:

1. Definiere einen geeigneten Linspace für den Zahlenraum von 0 bis 100. (Tipp: Achte darauf, dass die Quadratwurzel nur für nicht-negative Zahlen definiert ist.)
2. Berechne die Werte für die Quadratwurzel mithilfe der Funktion np.sqrt.
3. Plote das Ergebnis mit `plt.plot` und gebe den Plot mit `plt.show()` aus.

# Squarerooot



```
xs = np.linspace(0, 100, num=200)
ys = np.sqrt(xs)

plt.plot(xs, ys)
plt.show()
```

# Aufgabe Plotting - Multiplot

8 Punkte

In der nächsten Aufgabe willst du zwei Funktionen gleichzeitig plotten:  $f(x) = \sqrt{x}$ ;  $x \geq 0$  und  $g(x) = x^2$ .

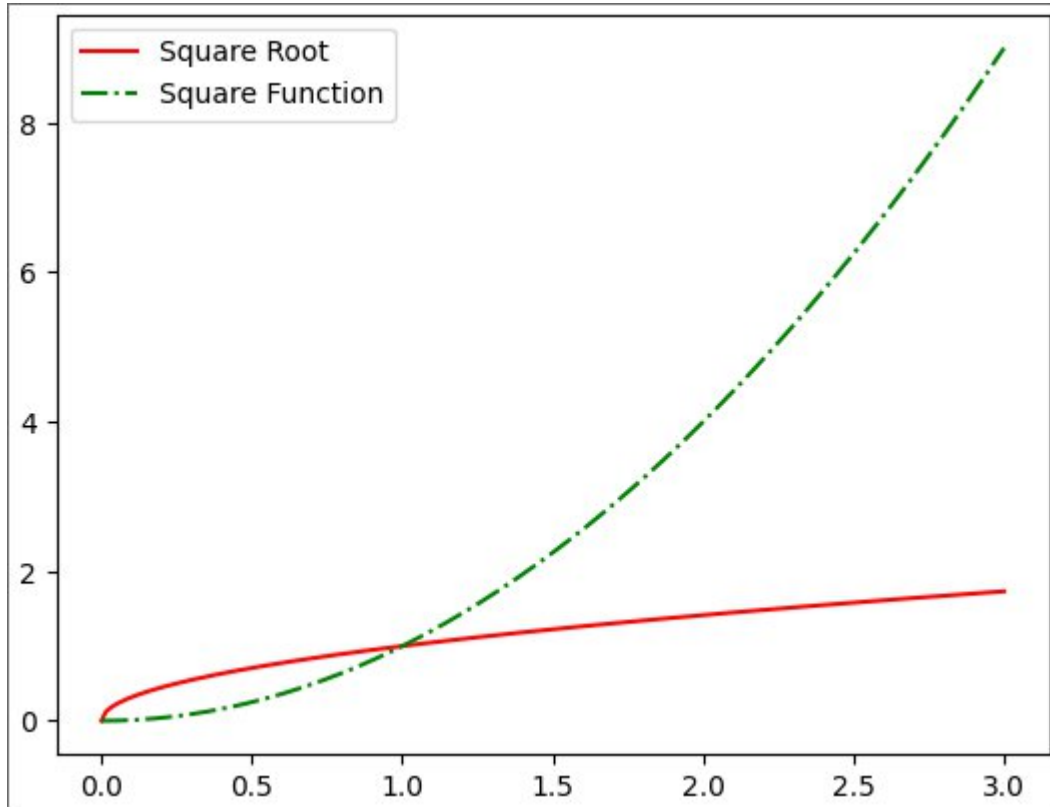
Gehe dabei wie folgt vor:

1. Definieren einen geeigneten Linspace für den Zahlenraum von 0 bis 3. (Tipp: Achte darauf, dass die Quadratwurzel nur für nicht-negative Zahlen definiert ist.)
2. Berechne die Werte für die Quadratwurzel mit np.sqrt.
3. Berechne die Werte für die Quadratzahlen mit np.square.
4. Gib den beiden Plots die Farben Grün und Rot. Nutze hierfür die Color Shorthands aus der Dokumentation.

# Aufgabe Plotting - Multiplot

5. Plote die Quadratfunktion mit dem Linestyle dashdot, wie in der Dokumentation zu [Linestyles](#) beschrieben.
6. Vergebe angemessene Labels für beide Plots.
7. Füge die Legende hinzu.
8. Plote das Ergebnis mit `plt.show()`.

# Multiplot



```
xt = np.linspace(0, 3, num=200)
sqrt = np.sqrt(xt)
square = np.square(xt)

plt.plot(xt, sqrt, color='r', label="Square Root")
plt.plot(xt, square, color='g', label="Square Function",
linestyle="dashdot")
plt.legend()
plt.show()
```

# Aufgabe – Pseudo Randomness

$$X_{n+1} = (a X_n + c) \bmod m, n \geq 0$$

Wenn  $c = 0$  gilt, spricht man auch vom **Multiplicative Congruent Generator (MCG)**.

Die einzelnen Werte haben folgende Bedeutung:

- **$X_n$**  ist der Startwert, auch **Seed** genannt.
- **$X_{n+1}$**  ist der Folgewert, der im nächsten Schritt als  **$X_n$**  verwendet wird.
- **$a$**  ist der Vorfaktor des Startwerts, wird auch als **Skalar** bezeichnet, da er den Wert skaliert.
- **$c$**  ist das hinzuaddierte **Offset**.
- **$m$**  ist der Restklassenring oder **Modulus**, der die Werte in einem festen Bereich hält.



# Aufgabe – Pseudo Randomness

6 Punkte

Schreibe einen **Linear Congruent Generator** mit dem Funktionsnamen `lcg`.

- Verwende die oben gegebene mathematische Definition
- Prüfe, dass die Eingabewerte gültig sind.
- `lcg` soll als unendlicher Generator implementiert werden, der bei jedem Aufruf den nächsten Wert der Pseudo-Zufallsfolge liefert.
- Ekläre in der Markdownzeile die Funktionsweise des Generators. (Keine Erklärung = 0 Punkte)

# Linear Congruent Generator

Python

```
def lcg(seed: int, scalar: int, modulus: int, offset: int) -> int:
    """
    Linear Congruential Generators

    
$$X(n+1) = (a X(n) + c) \bmod m; n \geq 0$$


    m > 0;
    0 <= a < m;
    c > 0; a > 0

    """
    assert modulus > 0, "Modulus must be greater than 0"
    assert 0 <= scalar and scalar < modulus, "Scalar must be in range 0 <= a < m"

    while seed > 1:
        seed = (scalar*seed+offset) % modulus
        assert seed >= 0
        yield seed
```

# Linear Congruent Generator - Funktionsweise

*Der Generator implementiert einen linearen Kongruenzgenerator (LCG), der aus einem Startwert seed durch die rekursive Formel (siehe Definition) eine Folge pseudozufälliger Zahlen erzeugt. Bei jedem Durchlauf wird der aktuelle Zustand mit dem Skalar  $a$ , dem Offset  $c$  und dem Modulus  $m$  zu einem neuen Wert transformiert, der anschließend ausgegeben wird.*

1. 3089810780120156248
2. 8356396685252565260
3. 1921117399837525548
4. 14806858147081821235
5. 2557599628047639428

```
# Test
rand = lcg(
    3_935_559_000_370_003_845,
    3_203_021_881_815_356_449,
    2**64-1,
    11_742_185_885_288_659_963
)

for i in range(5):
    print(f"{i+1}. {next(rand)}")
```

# Aufgabe – Plotting a PCG

6 Punkte

Plotte die Zufallszahlen eines `_Permuted Congruent Generators_` (PCG) mithilfe von NumPy und Matplotlib.

- Gegeben ist der Anfangszustand des Generators.
- Verwende die NumPy-Dokumentation um 20 Zufallszahlen zu erzeugen und speichere sie in der Variablen `pcgs` (Tipp: als NumPy Array).
- Sortiere anschließend die Werte in `pcgs` und speichere das Ergebnis in `pcgs_sorted`.
- Plotte sowohl die ursprünglichen als auch die sortierten Zufallszahlen in einem sinnvollen Diagramm (z. B. Linienplot oder Scatterplot) und gestalte den Plot übersichtlich.
- Erkläre in der gegebenen Markdown Zeile deinen Plot. (Keine Erklärung = 0 Punkte)

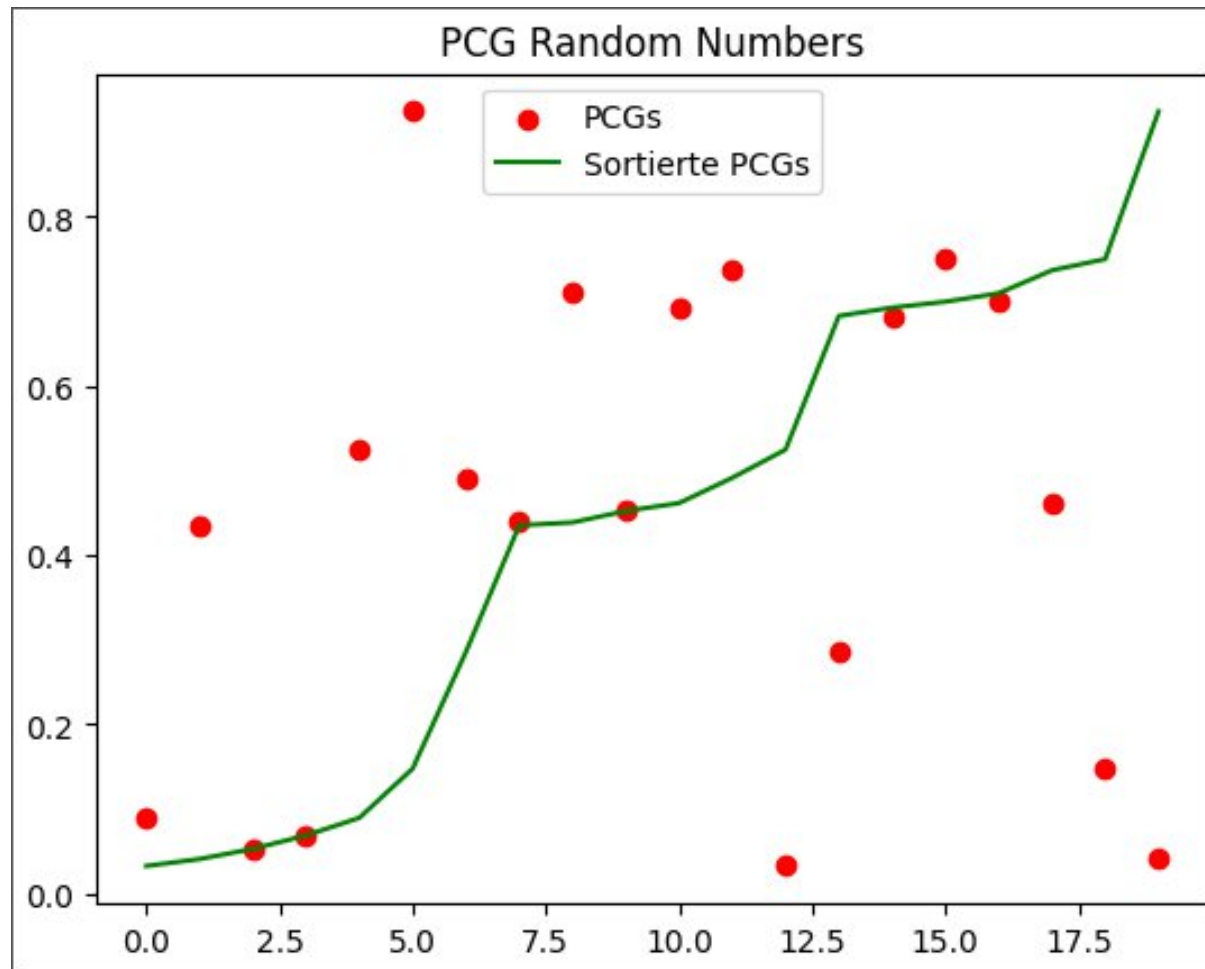
# Permuted Congruent Generator

```
Python
np.random.seed(42) # Setting a fixed start Value for the Generator
pcgs: np.array = None
pcgs_sorted: np.array = None

gen = np.random.default_rng()
pcgs = np.array([gen.random() for _ in range(20)])
pcgs_sorted = np.sort(pcgs)

# Plot
plt.scatter(np.arange(len(pcgs_sorted)), pcgs, color='r', label='PCGs')
plt.plot(np.arange(len(pcgs_sorted)), pcgs_sorted, color='g',
label='Sortierte PCGs')
plt.title("PCG Random Numbers")
plt.legend()
plt.show()
```

# Permuted Congruent Generator



*Zu sehen sind in Rot die mittels eines permuted congruential generators (PCG) erzeugten Zufallszahlen im Bereich von 0.0 bis 1.0; in Grün erscheinen dieselben Werte, jedoch sortiert und als Linie geplottet. Ein erkennbares Muster entsteht dabei nicht. Lediglich der sortierte Verlauf zeigt – rein rechnerisch bedingt – einen monoton steigenden Graphen, was für zufällige Werte in dieser Darstellung zu erwarten ist.*

# Aufgabe – Bar Plot

5 Punkte

Dir liegt ein Datenset `sec_school` einer Hauptschule vor, das die Klassenstufen von 5 bis 9 auf die Anzahl ihrer Schüler im jeweiligen Jahrgang abbildet.

Erstelle einen Barplot und gehe dabei wie folgt vor:

1. Wähle ein passendes Farbschema, um die einzelnen Klassenstufen anschaulich darzustellen.
2. Extrahiere die Schlüssel (Klassenstufen) und Werte (Anzahl der Schüler) aus dem Datenset und übergebe diese zusammen mit den Farbwerten an `plt.bar()`.
3. Setze geeignete Beschriftungen für die X- und Y-Achse, um die Darstellung verständlich zu machen.
4. Vergiss nicht, einen passenden Titel für den Plot zu setzen.
5. Gib den Plot mit `plt.show()` aus, um die Werte zu visualisieren.

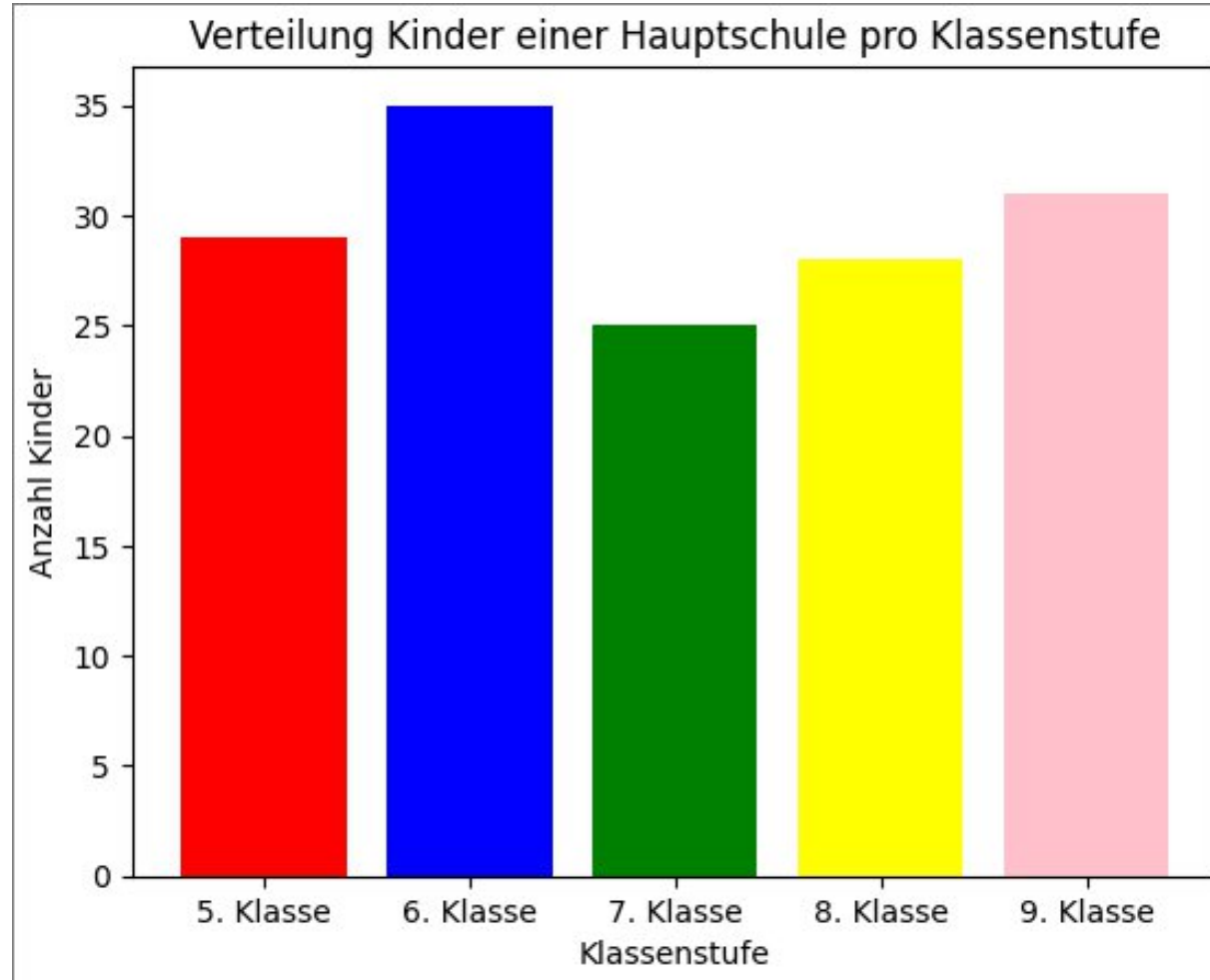


# Bar Plot - Hauptschule

```
sec_school = {  
    '5. Klasse': 29,  
    '6. Klasse': 35,  
    '7. Klasse': 25,  
    '8. Klasse': 28,  
    '9. Klasse': 31  
}  
  
colors = ["red", "blue", "green", "yellow", "pink"]  
  
plt.bar(sec_school.keys(), sec_school.values(), color=colors)  
plt.xlabel("Klassenstufe")  
plt.ylabel("Anzahl Kinder")  
plt.title("Verteilung Kinder einer Hauptschule pro Klassenstufe")  
plt.show()
```

Python

# Bar Plot - Hauptschule



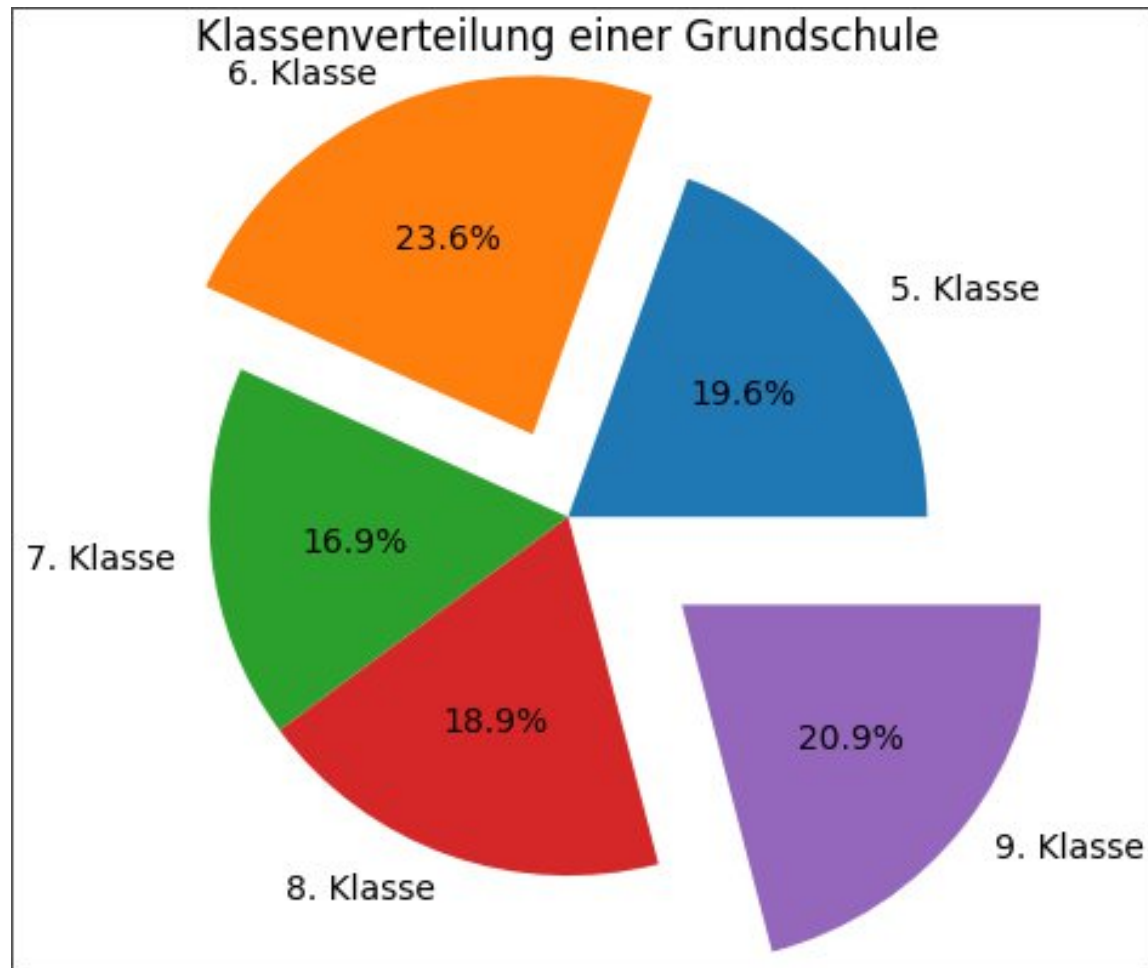
# Aufgabe – Pie Plot

5 Punkte

Erstelle einen Pieplot und gehe dabei wie folgt vor:

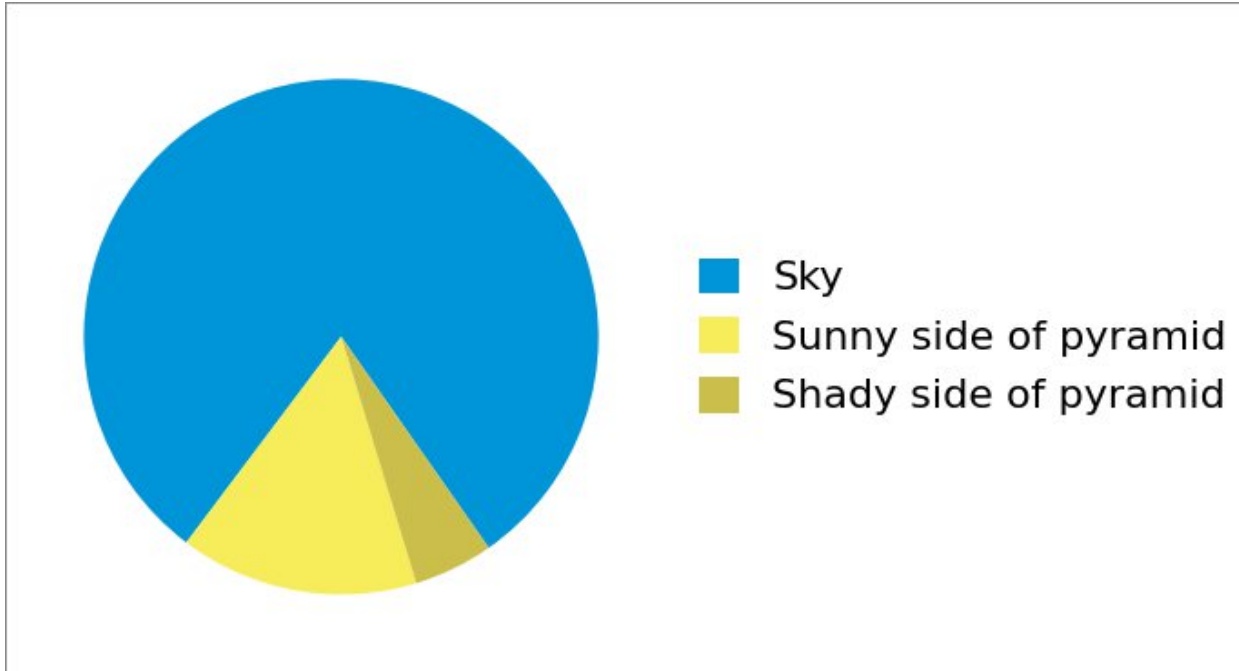
1. Wähle ein passendes Farbschema, um die einzelnen Klassenstufen anschaulich darzustellen.
2. Extrahiere die Schlüssel (Klassenstufen) und Werte (Anzahl der Schüler) aus dem Datenset und übergebe diese zusammen mit den Farbwerten an `plt.pie()`. Nutze `autopct='%1.1f%%'`, um die Prozentwerte der Segmente anzuzeigen.
3. Lasse die 6. Klasse um 25 % und die 9. Klasse um 40 % aus dem Kuchen hervortreten, indem du den Parameter `explode` entsprechend setzt.
4. Setze einen geeigneten Titel für den Plot.
5. Gib den Plot mit `plt.show()` aus, um die Werte zu visualisieren.

# Pie Plot - Hauptschule



```
sec_school = {  
    '5. Klasse': 29,  
    '6. Klasse': 35,  
    '7. Klasse': 25,  
    '8. Klasse': 28,  
    '9. Klasse': 31  
}  
  
plt.pie(  
    sec_school.values(),  
    labels=sec_school.keys(),  
    autopct='%1.1f%%',  
    explode=[0, 0.25, 0, 0, 0.4]  
)  
  
plt.title("Klassenverteilung einer Grundschule")  
  
plt.show()
```

# Bonus - Meme



```
from matplotlib.patches import Patch

colors = ['#0095D9', '#F7EC59', '#CBBE4A'] # Color Picker helps

plt.pie(
    [0.8, 0.15, 0.05], # Try and Error ratios
    colors=colors,
    startangle=-55
)

plt.legend(
    labels=["Sky", "Sunny side of pyramid", "Shady side of pyramid"],
    # Set right color and size for Patches
    handles=[Patch(facecolor=color) for color in colors],
    handlelength=1,
    handleheight=1,
    # Move Legend to the right
    loc="center left",
    bbox_to_anchor=(1, 0.5),
    fontsize=16,
    # Disable background and edge
    facecolor='none',
    edgecolor='none'
)

plt.tight_layout() # Layout shouldnt Overflow

plt.show()
```