Results

Einfache Aufgaben

1. **Begrüßungsfunktion**

```python
def begruessung(name):
    return f"Hallo, {name}!"
```

2. **Quadratfunktion**

```python
def quadrat(zahl):
    return zahl ** 2
```

3. **Listenumkehrung**

```python
def umkehren(liste):
    return liste[::-1]
```

4. **Maximum einer Liste**

```python
def maximum(liste):
    return max(liste)
```

5. **Zahlenreihe summieren**

```python
def summe(zahl):
    return sum(range(1, zahl + 1))
```

6. **Anzahl der Vokale**

```python
def anzahl_vokale(text):
    vokale = "aeiouAEIOU"
    return sum(1 for char in text if char in vokale)
```

Mittlere Aufgaben

1. **Primzahlprüfung**

```python
def ist_primzahl(n):
    if n <= 1:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

2. **Wörterzählung**

```python
def wortanzahl(text):
    woerter = text.split()
    zaehlung = {}
    for wort in woerter:
        if wort in zaehlung:
            zaehlung[wort] += 1
        else:
            zaehlung[wort] = 1
    return zaehlung
```

3. **Fibonacci-Sequenz**

```python
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

4. **Faktorialberechnung**

```python
def faktorial(n):
    if n == 0:
        return 1
    else:
        return n * faktorial(n - 1)
```

5. **Einzigartige Elemente**

```
def einzigartig(liste):
    return list(set(liste))
```

6. **Palindrome überprüfen**

```
def ist_palindrom(wort):
    return wort == wort[::-1]
```

Schwere Aufgaben

1. **Verschachtelte Listenabflachung**

```
def flatten(nested_list):
    flache_liste = []
    for item in nested_list:
        if isinstance(item, list):
            flache_liste.extend(flatten(item))
        else:
            flache_liste.append(item)
    return flache_liste
```

2. **Funktionsdekorator mit Argumenten**

```
def wiederhole(n):
    def dekorator(func):
        def wrapper(*args, **kwargs):
            ergebnisse = []
            for _ in range(n):
                ergebnisse.append(func(*args, **kwargs))
            return ergebnisse
        return wrapper
    return dekorator
```

3. **Asynchroner Web Scraper**

```
import aiohttp
from bs4 import BeautifulSoup
import asyncio
```

```python
async def fetch(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            return await response.text()

async def get_title(url):
    html = await fetch(url)
    soup = BeautifulSoup(html, "html.parser")
    return soup.title.string

async def async_scrape(urls):
    tasks = [get_title(url) for url in urls]
    return await asyncio.gather(*tasks)
```

4. **Longest Common Substring**

```python
def longest_common_substring(str1, str2):
    matrix = [[0] * (1 + len(str2)) for _ in range(1 + len(str1))]
    longest, x_longest = 0, 0
    for x in range(1, 1 + len(str1)):
        for y in range(1, 1 + len(str2)):
            if str1[x - 1] == str2[y - 1]:
                matrix[x][y] = matrix[x - 1][y - 1] + 1
                if matrix[x][y] > longest:
                    longest = matrix[x][y]
                    x_longest = x
            else:
                matrix[x][y] = 0
    return str1[x_longest - longest: x_longest]
```

5. **Tiefensuche in einem Graphen**

```python
def tiefensuche(graph, start, besucht=None):
    if besucht is None:
        besucht = []
    besucht.append(start)
    for nachbar in graph[start]:
        if nachbar not in besucht:
            tiefensuche(graph, nachbar, besucht)
    return besucht
```

6. **Zahlen in Wörter umwandeln**

```
def zahl_in_wort(n):
    einheiten = ["", "eins", "zwei", "drei", "vier", "fünf", "sechs",
"sieben", "acht", "neun"]
    zehner = ["", "zehn", "zwanzig", "dreißig", "vierzig", "fünfzig",
"sechzig", "siebzig", "achtzig", "neunzig"]
    besondere_zehner = {11: "elf", 12: "zwölf"}
    hunderter = "hundert"

    if n == 0:
        return "null"
    elif n < 10:
        return einheiten[n]
    elif n < 20:
        return besondere_zehner.get(n, einheiten[n % 10] + "zehn")
    elif n < 100:
        return einheiten[n % 10] + "und" + zehner[n // 10]
    else:
        return einheiten[n // 100] + hunderter + (zahl_in_wort(n % 100) if
n % 100 != 0 else "")
```