Aufgaben

Einfache Aufgaben

1. **Begrüßungsfunktion**
   Schreiben Sie eine Funktion `begruessung(name)`, die einen Namen als Parameter nimmt und eine Begrüßung ausgibt.

   Beispiel:

   ```
   >>> begruessung("Anna")
   "Hallo, Anna!"
   ```

2. **Quadratfunktion**
   Erstellen Sie eine Funktion `quadrat(zahl)`, die das Quadrat einer gegebenen Zahl zurückgibt.

   Beispiel:

   ```
   >>> quadrat(4)
   16
   ```

3. **Listenumkehrung**
   Schreiben Sie eine Funktion `umkehren(liste)`, die eine Liste als Parameter nimmt und eine neue Liste mit den Elementen in umgekehrter Reihenfolge zurückgibt.

   Beispiel:

   ```
   >>> umkehren([1, 2, 3, 4])
   [4, 3, 2, 1]
   ```

4. **Maximum einer Liste**
   Schreiben Sie eine Funktion `maximum(liste)`, die das größte Element in einer Liste zurückgibt.

   Beispiel:

```
>>> maximum([1, 5, 3, 9])
9
```

5. **Zahlenreihe summieren**
   Implementieren Sie eine Funktion `summe(zahl)`, die die Summe der Zahlen von 1 bis zur gegebenen Zahl berechnet.

   Beispiel:

```
>>> summe(5)
15  # (1 + 2 + 3 + 4 + 5)
```

6. **Anzahl der Vokale**
   Schreiben Sie eine Funktion `anzahl_vokale(text)`, die die Anzahl der Vokale in einem gegebenen Text zählt.

   Beispiel:

```
>>> anzahl_vokale("Hallo Welt")
3
```

Mittlere Aufgaben

1. **Primzahlprüfung**
   Implementieren Sie eine Funktion `ist_primzahl(n)`, die überprüft, ob eine gegebene Zahl n eine Primzahl ist. Die Funktion soll True zurückgeben, wenn n eine Primzahl ist, und False andernfalls.

   Beispiel:

```
>>> ist_primzahl(17)
True
>>> ist_primzahl(24)
False
```

2. **Wörterzählung**
   Schreiben Sie eine Funktion `wortanzahl(text)`, die einen String als Input nimmt und ein Dictionary zurückgibt, das die Häufigkeit jedes Wortes im Text enthält.

Beispiel:

```
>>> wortanzahl("Der Hund läuft. Der Hund bellt.")
{'Der': 2, 'Hund': 2, 'läuft': 1, 'bellt': 1}
```

3. **Fibonacci-Sequenz**
   Erstellen Sie eine Funktion `fibonacci(n)`, die das n-te Element der Fibonacci-Sequenz zurückgibt. Verwenden Sie Rekursion für diese Aufgabe.

   Beispiel:

```
>>> fibonacci(7)
13
```

4. **Faktorialberechnung**
   Implementieren Sie eine Funktion `faktorial(n)`, die das Fakultät einer gegebenen Zahl berechnet.

   Beispiel:

```
>>> faktorial(5)
120  # (5 * 4 * 3 * 2 * 1)
```

5. **Einzigartige Elemente**
   Schreiben Sie eine Funktion `einzigartig(liste)`, die eine Liste nimmt und die Liste ohne Duplikate zurückgibt.

   Beispiel:

```
>>> einzigartig([1, 2, 2, 3, 4, 4, 5])
[1, 2, 3, 4, 5]
```

6. **Palindrome überprüfen**
   Schreiben Sie eine Funktion `ist_palindrom(wort)`, die überprüft, ob ein Wort ein Palindrom ist (vorwärts und rückwärts gleich).

   Beispiel:

```
>>> ist_palindrom("otto")
True
```

```
>>> ist_palindrom("python")
False
```

Schwere Aufgaben

1. **Verschachtelte Listenabflachung**
   Implementieren Sie eine Funktion `flatten(nested_list)`, die eine verschachtelte Liste beliebiger Tiefe als Input nimmt und eine flache Liste zurückgibt.

   Beispiel:

   ```
   >>> flatten([1, [2, [3, 4], 5], 6])
   [1, 2, 3, 4, 5, 6]
   ```

2. **Funktionsdekorator mit Argumenten**
   Schreiben Sie einen Dekorator `wiederhole(n)`, der eine Funktion n-mal ausführt und die Ergebnisse in einer Liste zurückgibt. Der Dekorator soll mit beliebigen Funktionen funktionieren.

   Beispiel:

   ```
   @wiederhole(3)
   def gruss(name):
       return f"Hallo, {name}!"

   >>> gruss("Max")
   ["Hallo, Max!", "Hallo, Max!", "Hallo, Max!"]
   ```

3. **Asynchroner Web Scraper**
   Erstellen Sie eine asynchrone Funktion `async_scrape(urls)`, die eine Liste von URLs als Input nimmt und parallel den Titel jeder Webseite extrahiert. Verwenden Sie die `aiohttp` und `beautifulsoup4` Bibliotheken.

   Beispiel:

   ```
   urls = [
       "https://www.python.org",
       "https://www.github.com",
       "https://www.stackoverflow.com"
   ]
   >>> asyncio.run(async_scrape(urls))
   ```

```
[
    "Welcome to Python.org",
    "GitHub: Where the world builds software · GitHub",
    "Stack Overflow - Where Developers Learn, Share, & Build Careers"
]
```

4. **Longest Common Substring**
   Schreiben Sie eine Funktion `longest_common_substring(str1, str2)`, die den längsten gemeinsamen Substring von zwei gegebenen Strings berechnet.

   Beispiel:

```
>>> longest_common_substring("abcdef", "zabcf")
"abc"
```

5. **Tiefensuche in einem Graphen**
   Implementieren Sie eine Funktion `tiefensuche(graph, start)`, die die Tiefensuche (DFS) in einem gegebenen Graphen durchführt.

   Beispiel:

```
>>> graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}
>>> tiefensuche(graph, 'A')
['A', 'B', 'D', 'E', 'F', 'C']
```

6. **Zahlen in Wörter umwandeln**
   Schreiben Sie eine Funktion `zahl_in_wort(n)`, die eine Zahl von 0 bis 999 in deutsche Wörter umwandelt.

   Beispiel:

```
>>> zahl_in_wort(342)
"dreihundertzweiundvierzig"
```