Technische Eigenschaften der Programmiersprache Python

Einfachheit und Lesbarkeit

- Python betont eine klare, saubere Syntax, die es Entwicklern ermöglicht, Code leicht zu verstehen und zu schreiben.
- Der Code ist oft wie Pseudocode, was das Lesen und Schreiben von Python-Code erleichtert.

Beispiel:

In [9]:
```python
# Ein einfaches Python-Beispiel
def gruß(name):
    print("Hallo, " + name + "!")

gruß("Python-Entwickler")
```

```
Hallo, Python-Entwickler!
```

Vielseitigkeit und Modularität

- Python ist eine vielseitige Programmiersprache, die in verschiedenen Anwendungsdomänen eingesetzt werden kann, darunter
  - Webentwicklung
  - wissenschaftliche Berechnungen
  - Datenanalyse.
- Python fördert die Modulbildung und ermöglicht die Erstellung wiederverwendbarer Codekomponenten.

Beispiel 1:

In [19]:
```python
# berechnungen.py

def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

def divide(a, b):
    if b == 0:
        return "Division durch Null ist nicht erlaubt."
    return a / b
```

In [21]:
```python
#import berechnungen  # Importieren des berechnungen-Moduls

# Verwenden der Funktionen aus dem Modul
result1 = add(5, 3)
result2 = subtract(10, 4)
result3 = multiply(6, 7)
result4 = divide(8, 2)

# Ausgabe der Ergebnisse
print("Addition:", result1)
print("Subtraktion:", result2)
print("Multiplikation:", result3)
print("Division:", result4)
```

```
Addition: 8
Subtraktion: 6
Multiplikation: 42
Division: 4.0
```

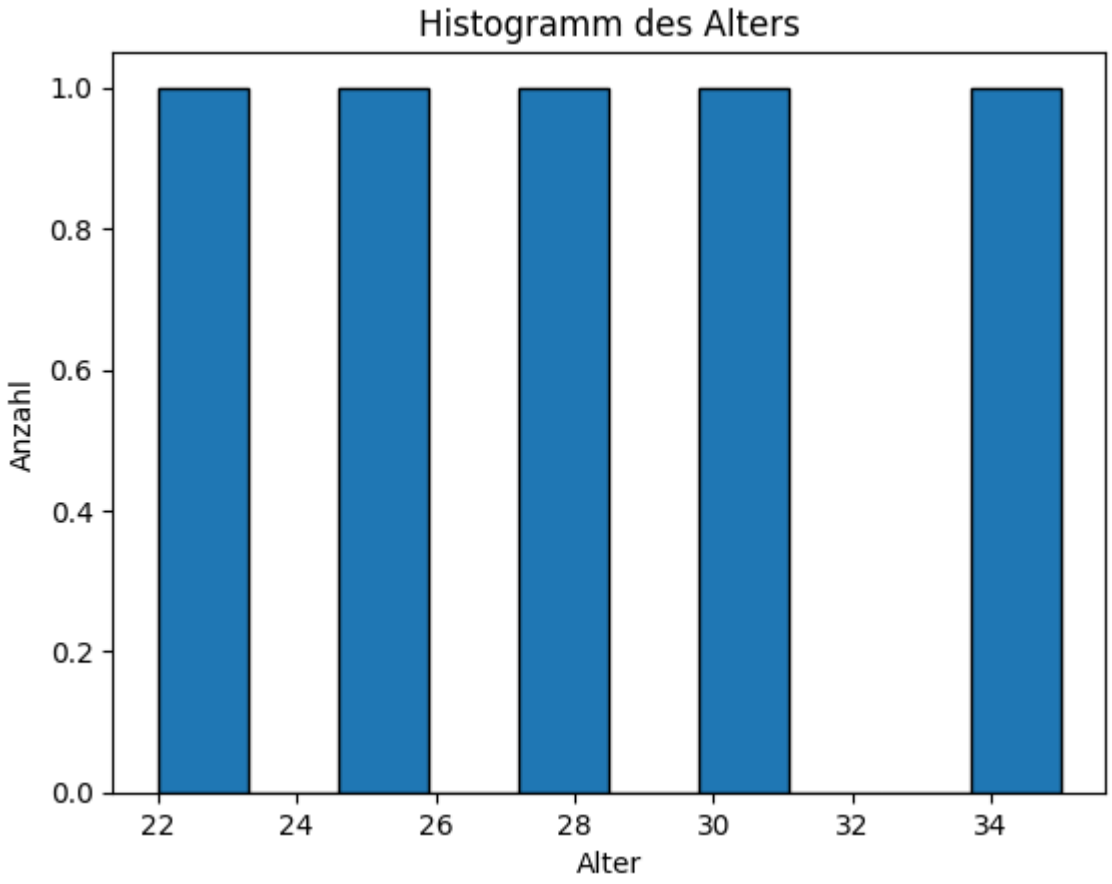
In [10]:
```python
# Ein Beispiel für die Verwendung eines Moduls in Python
import math

radius = 5
fläche = math.pi * radius**2
print("Die Fläche des Kreises beträgt:", fläche)
```

```
Die Fläche des Kreises beträgt: 78.53981633974483
```

In [18]:
```python
import pandas as pd
import matplotlib.pyplot as plt

# Generieren von Beispiel-Daten
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
    'Alter': [25, 30, 35, 28, 22],
    'Einkommen': [50000, 60000, 75000, 48000, 52000],
    'Stadt': ['Berlin', 'München', 'Hamburg', 'Berlin', 'München']
}

# Erstellen eines Pandas-Datenframes
df = pd.DataFrame(data)

# Datenüberblick
print("Datenüberblick:")
print(df)

# Statistische Zusammenfassung der Daten
print("\nStatistische Zusammenfassung der Daten:")
print(df.describe())

# Gruppieren und Aggregieren von Daten
nach_stadt = df.groupby('Stadt')['Einkommen'].mean()
print("\nDurchschnittliches Einkommen nach Stadt:")
print(nach_stadt)

# Datenvisualisierung: Histogramm des Alters
```

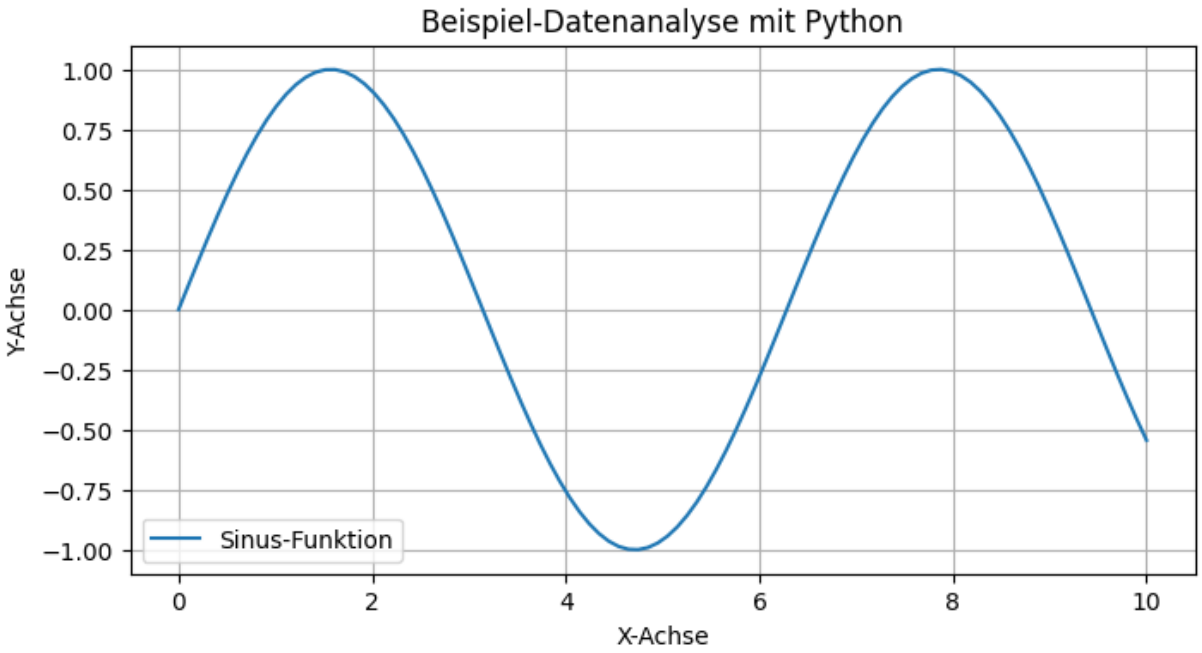
```
plt.hist(df['Alter'], bins=10, edgecolor='k')
plt.title('Histogramm des Alters')
plt.xlabel('Alter')
plt.ylabel('Anzahl')
plt.show()
```

```
Datenüberblick:
      Name  Alter  Einkommen    Stadt
0    Alice     25      50000   Berlin
1      Bob     30      60000  München
2  Charlie     35      75000  Hamburg
3    David     28      48000   Berlin
4      Eva     22      52000  München

Statistische Zusammenfassung der Daten:
           Alter     Einkommen
count   5.000000      5.000000
mean   28.000000  57000.000000
std     4.949747  11045.361017
min    22.000000  48000.000000
25%    25.000000  50000.000000
50%    28.000000  52000.000000
75%    30.000000  60000.000000
max    35.000000  75000.000000

Durchschnittliches Einkommen nach Stadt:
Stadt
Berlin     49000.0
Hamburg    75000.0
München    56000.0
Name: Einkommen, dtype: float64
```



Beispiel 2:

In [ ]:
```python
# Importieren der benötigten Bibliotheken
import numpy as np
import matplotlib.pyplot as plt

# Generieren von Beispiel-Daten
x = np.linspace(0, 10, 100)  # Erzeugt eine NumPy-Array mit 100 Werten von 0
y = np.sin(x)  # Berechnet den Sinus für jeden Wert in x

# Datenvisualisierung mit Matplotlib
plt.figure(figsize=(8, 4))  # Erstellt eine Figure mit einer bestimmten Größ
plt.plot(x, y, label='Sinus-Funktion')  # Plotten der Sinus-Funktion
plt.title('Beispiel-Datenanalyse mit Python')  # Titel des Diagramms
plt.xlabel('X-Achse')  # Beschriftung der X-Achse
plt.ylabel('Y-Achse')  # Beschriftung der Y-Achse
plt.legend()  # Anzeige der Legende
plt.grid(True)  # Raster anzeigen
plt.show()  # Anzeigen des Diagramms
```



Unterstützung der Community

- Python verfügt über eine lebhafte und engagierte Entwicklergemeinschaft, die zur kontinuierlichen Verbesserung der Sprache und zur Erstellung von Erweiterungen und Bibliotheken beiträgt.
- Die Python Software Foundation (PSF) fördert die Entwicklung und Wartung von Python und bietet finanzielle Unterstützung für Projekte.

Beispiel:

In [17]:
```python
# Ein Beispiel für die Verwendung einer Community-Bibliothek
import requests

url = "https://www.python.org"
response = requests.get(url)
print("HTTP-Statuscode:", response.status_code)
```

```
HTTP-Statuscode: 200
```

Plattformunabhängigkeit und Open Source

- Python ist eine Interpretierte Programmiersprache.
- Python ist plattformunabhängig und kann auf verschiedenen Betriebssystemen ohne Änderungen am Code ausgeführt werden.
- Python ist eine Open-Source-Programmiersprache, bei der der Quellcode für alle frei verfügbar ist und von der Gemeinschaft gepflegt wird.

Beispiel:

In [16]:
```
import os

plattform = os.name
print("Die aktuelle Plattform ist:", plattform)
```

```
Die aktuelle Plattform ist: posix
```

Leistungsfähige Bibliotheken

- Python bietet eine umfangreiche Sammlung von Bibliotheken und Frameworks, die Entwicklern helfen, komplexe Aufgaben in verschiedenen Bereichen zu bewältigen.
- Diese Bibliotheken umfassen NumPy und SciPy für wissenschaftliche Berechnungen, pandas für Datenanalyse und TensorFlow für maschinelles Lernen.

Beispiel:

In [ ]:
```
import numpy as np

data = np.array([1, 2, 3, 4, 5])
durchschnitt = np.mean(data)
print("Durchschnitt:", durchschnitt)
```

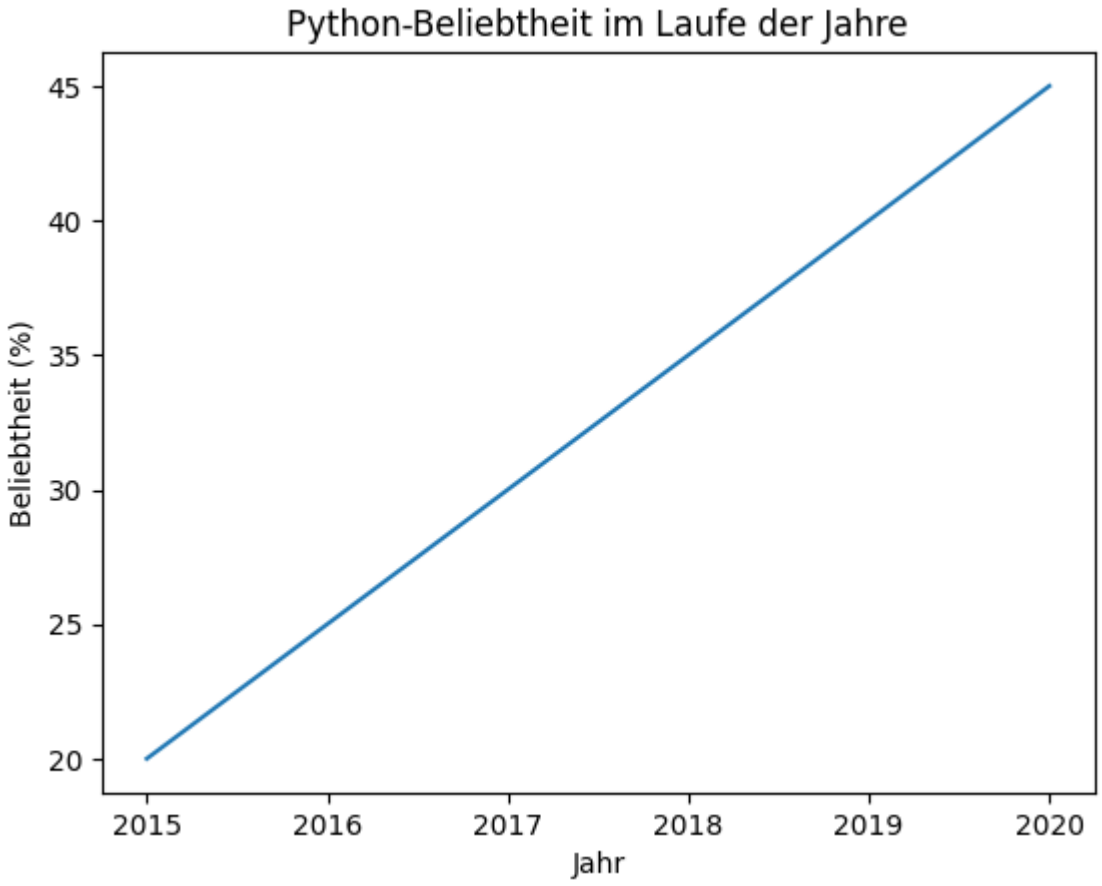
```
Durchschnitt: 3.0
```

Zukunftssicherheit und Beliebtheit

- Python ist eine der am häufigsten verwendeten Programmiersprachen weltweit und wird in verschiedenen Industrien und Anwendungsbereichen eingesetzt.
- Die kontinuierliche Weiterentwicklung von Python und die breite Akzeptanz machen sie zu einer zukunftssicheren Wahl für Entwickler.

Beispiel:

In [ ]:
```
# Ein Beispiel für die Verwendung von Statistiken zur Beliebtheit von Python
import matplotlib.pyplot as plt

jahre = [2015, 2016, 2017, 2018, 2019, 2020]
beliebtheit = [20, 25, 30, 35, 40, 45]

plt.plot(jahre, beliebtheit)
plt.xlabel("Jahr")
plt.ylabel("Beliebtheit (%)")
plt.title("Python-Beliebtheit im Laufe der Jahre")
plt.show()
```



Python ist objektorientert

- Die Objektorientierung in Python ermöglicht die Modellierung realer oder abstrakter Konzepte in Ihrem Code auf eine organisierte und wiederverwendbare Weise.
- Objektorientierte Programme sind oft leichter zu verstehen und zu warten, da sie das Prinzip der Modularität und Wiederverwendbarkeit fördern.

1. **Klassen und Objekte:** Python ermöglicht die Definition von Klassen, die als Baupläne dienen, um Objekte zu erstellen. Eine Klasse definiert die Struktur und das Verhalten von Objekten, während Objekte spezifische Instanzen dieser Klasse sind.

2. **Attribute und Methoden:** Klassen können Attribute (Daten) und Methoden (Funktionen) enthalten. Attribute repräsentieren Eigenschaften eines Objekts, während Methoden Operationen ausführen, die mit dem Objekt verknüpft sind.

3. **Konstruktor:** Python-Klassen können einen Konstruktor (**init**) haben, der zur Initialisierung von Attributen verwendet wird, wenn ein Objekt erstellt wird.

4. **Vererbung:** Python unterstützt die Vererbung, wodurch Sie eine neue Klasse von einer bestehenden Klasse ableiten können. Die abgeleitete Klasse erbt Attribute und Methoden von der Basisklasse und kann sie erweitern oder überschreiben.

5. **Polymorphismus:** Python ermöglicht Polymorphismus, was bedeutet, dass verschiedene Klassen Objekte mit denselben Methodennamen haben können. Dies erlaubt es, auf unterschiedliche Objekte einheitlich zuzugreifen.

6. **Kapselung:** Python verwendet Konventionen wie "Private" und "Öffentlich", um die Sichtbarkeit von Attributen und Methoden in Klassen zu steuern. Obwohl Python keine strikte Kapselung wie einige andere Sprachen hat, wird die Verwendung von Unterstrichen (z.B. _attribut) als Konvention verwendet, um auf private Attribute hinzuweisen.

7. **Abstraktion:** Python erlaubt die Erstellung abstrakter Klassen, die nicht direkt instanziiert werden können, sondern als Vorlage für abgeleitete Klassen dienen.

8. **Duck Typing:** Python verwendet Duck Typing, bei dem der Typ eines Objekts anhand seiner Eigenschaften und Methoden während der Laufzeit bestimmt wird. Dies ermöglicht eine flexible Programmierung.

In [23]:
```python
# Definition der Klasse "Person"
class Person:
    # Konstruktor-Methode zur Initialisierung von Attributen
    def __init__(self, vorname, nachname, alter):
        self.vorname = vorname
        self.nachname = nachname
        self.alter = alter

    # Methode zur Anzeige von Informationen über die Person
    def info_anzeigen(self):
        print(f"Name: {self.vorname} {self.nachname}")
        print(f"Alter: {self.alter} Jahre")

# Erstellung von Objekten (Instanzen) der Klasse "Person"
person1 = Person("Max", "Mustermann", 30)
person2 = Person("Anna", "Musterfrau", 25)

# Aufruf der Methode zur Anzeige von Informationen über die Personen
print("Person 1:")
person1.info_anzeigen()

print("\nPerson 2:")
person2.info_anzeigen()
```

```
Person 1:
Name: Max Mustermann
Alter: 30 Jahre

Person 2:
Name: Anna Musterfrau
Alter: 25 Jahre
```

In [ ]: