Einführung

Eine kurze Einführung in Python

Python ist eine Programmiersprache, die von Guido van Rossum in den späten 1980er Jahren entwickelt wurde. Python ist einfach zu lernen und zu lesen, was es zu einer beliebten Sprache für Anfänger und Experten gleichermaßen macht. Python wird in vielen Bereichen eingesetzt, einschließlich Webentwicklung, Datenanalyse, künstlicher Intelligenz und wissenschaftlicher Berechnungen.

Grundlagen

Das ist ein einfaches Python Programm

```
# Grundlagen von Python

# Kommentare können mit einem Hashtag (#) beginnen und werden vom
Interpreter ignoriert
# Kommentare sind nützlich, um den Code zu dokumentieren und zu
erklären, was er tut

# Hier ist ein Beispiel für einen Kommentar
# Dieser Code gibt "Hallo Welt!" aus
print("Hallo Welt!")
```

Variablen und Datentypen

Wie man Variablen in Python deklariert und welche Datentypen es gibt.

```
# Variablen und Datentypen

# Variablen sind Container, die Werte speichern können
# In Python müssen Variablen nicht deklariert werden, sie werden
automatisch erstellt, wenn ihnen ein Wert zugewiesen wird

# Hier ist ein Beispiel für eine Variable
x = 5
print(x)

# In Python gibt es verschiedene Datentypen, darunter:
# - Integer (ganze Zahlen)
# - Float (Dezimalzahlen)
```

```
# - String (Zeichenketten)
# - Boolean (Wahrheitswerte True oder False)

# Hier sind Beispiele für jeden Datentyp
a = 42 # Integer
b = 3.14 # Float
c = "Hallo" # String
d = True # Boolean

# Man kann den Datentyp einer Variable mit der Funktion type()
überprüfen
print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

Mathematik und Operationen

Wie man Mathematik und Operationen benutzt

```
#Operationen ermögliche es uns in Python berechnungen durchzuführen
#dafür gibt es verschiedene Operatoren, die man verwenden kann

# Grundlegende mathematische Operationen
a = 10
b = 5

# Addition
sum_result = a + b
print("Addition:", sum_result)

# Subtraktion
subtraction_result = a - b
print("Subtraktion:", subtraction_result)

# Multiplikation
multiplication_result = a * b
print("Multiplikation:", multiplication_result)

# Division (Gleitkommadivision)
division_result = a / b
print("Division:", division_result)

# Ganzzahldivision (abrunden)
integer_division_result = a // b
print("Ganzzahldivision:", integer_division_result)
```

```
# Modulo (Rest bei der Division)
modulo_result = a % b
print("Modulo:", modulo_result)

# Potenzierung
power_result = a ** b
print("Potenzierung:", power_result)
```

Bedingungen

Wie man Bedingungen in Python schreibt, einschließlich if-else-Anweisungen.

```
# Bedingungen

# Bedingungen ermöglichen es uns, Code auszuführen, wenn eine
bestimmte Bedingung erfüllt ist
# Die grundlegende Syntax für eine Bedingung ist "if", gefolgt von der
Bedingung und einem Doppelpunkt
# Der Code, der ausgeführt werden soll, wenn die Bedingung wahr ist,
wird eingerückt
#es gibt unterschiedliche Bedingungszeichen:
# == (gleich)
# != (ungleich)
# < (kleiner)
# > (größer)
# <= (kleiner gleich)
# >= (größer gleich)
#
#und noch ein paar mehr

# Hier ist ein Beispiel für eine Bedingung
if 5 > 2:
    print("Fünf ist größer als zwei")

# Man kann auch eine "else"-Anweisung hinzufügen, die ausgeführt wird,
wenn die Bedingung falsch ist
# Hier ist ein Beispiel für eine Bedingung mit "else"
if 5 < 2:
    print("Fünf ist kleiner als zwei")
else:
    print("Fünf ist nicht kleiner als zwei")

# Man kann auch mehrere Bedingungen mit "elif" hinzufügen
# Hier ist ein Beispiel für eine Bedingung mit "elif"
x = 5
```

```python
if x < 0:
    print("x ist negativ")
elif x == 0:
    print("x ist null")
else:
    print("x ist positiv")
```

Funktionen

Wie man Funktionen in Python definiert und aufruft.

```python
# Funktionen ermöglichen es uns, Code zu organisieren und wiederverwendbar zu machen
# Eine Funktion wird definiert mit dem Schlüsselwort "def", gefolgt vom Funktionsnamen, den Argumenten in Klammern und einem Doppelpunkt
# Der Code, der in der Funktion ausgeführt werden soll, wird eingerückt
# Eine Funktion kann einen Wert zurückgeben mit dem Schlüsselwort "return"

# Hier ist ein Beispiel für eine Funktion, die die Summe von zwei Zahlen berechnet
def addiere(zahl1, zahl2):
    summe = zahl1 + zahl2
    return summe

# Man ruft eine Funktion auf, indem man ihren Namen und die Argumente in Klammern angibt
# Hier ist ein Beispiel für den Aufruf der Funktion addiere
ergebnis = addiere(3, 5)
print(ergebnis) # gibt 8 aus
```

Listen

Wie man Listen in Python erstellt und bearbeitet.

```python
# Listen sind eine Sammlung von Elementen, die in einer bestimmten Reihenfolge angeordnet sind
# Listen werden mit eckigen Klammern erstellt und die Elemente werden durch Kommas getrennt

# Hier ist ein Beispiel für eine Liste von Zahlen
```

```python
zahlen = [1, 2, 3, 4, 5]
print(zahlen)

# Man kann auch eine Liste von Strings erstellen
fruits = ["Apfel", "Banane", "Kirsche"]
print(fruits)

# Man kann auf die Elemente einer Liste zugreifen, indem man ihren
Index angibt
# Der Index beginnt bei 0 für das erste Element und erhöht sich um 1
für jedes weitere Element
print(fruits[0]) # gibt "Apfel" aus
print(fruits[1]) # gibt "Banane" aus
print(fruits[2]) # gibt "Kirsche" aus

# Man kann auch auf die Elemente einer Liste von hinten zugreifen,
indem man negative Indizes verwendet
# Der Index -1 gibt das letzte Element zurück, -2 das vorletzte usw.
print(fruits[-1]) # gibt "Kirsche" aus
print(fruits[-2]) # gibt "Banane" aus
print(fruits[-3]) # gibt "Apfel" aus

# Man kann die Elemente einer Liste ändern, indem man ihren Index
angibt und einen neuen Wert zuweist
fruits[1] = "Orange"
print(fruits) # gibt ["Apfel", "Orange", "Kirsche"] aus

# Man kann auch Elemente zu einer Liste hinzufügen, indem man die
Methode append() verwendet
fruits.append("Erdbeere")
print(fruits) # gibt ["Apfel", "Orange", "Kirsche", "Erdbeere"] aus

# Man kann auch Elemente aus einer Liste entfernen, indem man die
Methode remove() verwendet
fruits.remove("Orange")
print(fruits) # gibt ["Apfel", "Kirsche", "Erdbeere"] aus
```

Dictionaries

Wie man Dictionaries in Python erstellt und bearbeitet.

```python
# Dictionaries sind eine Sammlung von Schlüssel-Wert-Paaren
# Jeder Schlüssel muss eindeutig sein und auf einen Wert verweisen
# Dictionaries werden mit geschweiften Klammern erstellt und die
Schlüssel-Wert-Paare werden durch Kommas getrennt
# Der Schlüssel und der Wert werden durch einen Doppelpunkt getrennt
```

```python
# Hier ist ein Beispiel für ein Dictionary von Personen
personen = {"Max": 25, "Anna": 30, "Peter": 40}
print(personen)

# Man kann auf den Wert eines Schlüssels zugreifen, indem man den Schlüssel angibt
print(personen["Max"]) # gibt 25 aus

# Man kann auch den Wert eines Schlüssels ändern, indem man den Schlüssel angibt und einen neuen Wert zuweist
personen["Max"] = 26
print(personen) # gibt {"Max": 26, "Anna": 30, "Peter": 40} aus

# Man kann auch ein neues Schlüssel-Wert-Paar hinzufügen, indem man den Schlüssel angibt und einen neuen Wert zuweist
personen["Lisa"] = 35
print(personen) # gibt {"Max": 26, "Anna": 30, "Peter": 40, "Lisa": 35} aus

# Man kann auch ein Schlüssel-Wert-Paar aus einem Dictionary entfernen, indem man den Schlüssel angibt
del personen["Peter"]
print(personen) # gibt {"Max": 26, "Anna": 30, "Lisa": 35} aus
```

Schleifen

Wie man Schleifen in Python schreibt, einschließlich for- und while-Schleifen.

```python
# Schleifen

# Schleifen ermöglichen es uns, Code mehrmals auszuführen
# Es gibt zwei Arten von Schleifen in Python: for-Schleifen und while-Schleifen

# Eine for-Schleife wird verwendet, um über eine Sequenz von Elementen zu iterieren
# Die grundlegende Syntax für eine for-Schleife ist "for", gefolgt von einer Variable, dem Schlüsselwort "in", der Sequenz und einem Doppelpunkt
# Der Code, der in jeder Iteration ausgeführt werden soll, wird eingerückt

# Hier ist ein Beispiel für eine for-Schleife
fruits = ["Apfel", "Banane", "Kirsche"]
for fruit in fruits:
```

```
    print(fruit)

# Eine while-Schleife wird verwendet, um Code auszuführen, solange
eine Bedingung wahr ist
# Die grundlegende Syntax für eine while-Schleife ist "while", gefolgt
von der Bedingung und einem Doppelpunkt
# Der Code, der in jeder Iteration ausgeführt werden soll, wird
eingerückt

# Hier ist ein Beispiel für eine while-Schleife
i = 1
while i < 6:
    print(i)
    i += 1

# Achte darauf, dass die Bedingung irgendwann falsch wird, sonst wird
die Schleife unendlich ausgeführt
```

Module und Pakete

Wie man Module und Pakete in Python importiert und verwendet.

```
# Module und Pakete

# In Python können wir Code in Module und Pakete organisieren, um ihn
wiederverwendbar zu machen
# Ein Modul ist eine Datei mit Python-Code, die Funktionen, Klassen
und Variablen enthält
# Ein Paket ist ein Verzeichnis, das Module und andere Pakete
enthalten kann

# Um ein Modul in Python zu importieren, verwenden wir das
Schlüsselwort "import", gefolgt vom Modulnamen
# Hier ist ein Beispiel für den Import des Moduls "math"
import math

# Wir können nun auf die Funktionen und Variablen des Moduls "math"
zugreifen
print(math.pi) # gibt die Kreiszahl Pi aus

# Wir können auch nur bestimmte Funktionen oder Variablen aus einem
Modul importieren, indem wir sie mit Kommas getrennt angeben
from math import sqrt, pow

# Wir können nun auf die Funktionen "sqrt" und "pow" zugreifen, ohne
den Modulnamen "math" zu verwenden
```

```
print(sqrt(16)) # gibt 4.0 aus
print(pow(2, 3)) # gibt 8.0 aus

# Wenn wir ein Modul mit einem anderen Namen importieren möchten,
können wir es mit dem Schlüsselwort "as" umbenennen
import math as m
print(m.pi) # gibt die Kreiszahl Pi aus

# Um ein Paket in Python zu importieren, verwenden wir das
Schlüsselwort "import", gefolgt vom Paketnamen
# Hier ist ein Beispiel für den Import des Pakets "os"
import os

# Wir können nun auf die Funktionen und Variablen des Pakets "os"
zugreifen
print(os.getcwd()) # gibt das aktuelle Arbeitsverzeichnis aus

# Wir können auch nur bestimmte Module aus einem Paket importieren,
indem wir sie mit Kommas getrennt angeben
from os import path, listdir

# Wir können nun auf die Module "path" und "listdir" zugreifen, ohne
den Paketnamen "os" zu verwenden
print(path.abspath("test.txt")) # gibt den absoluten Pfad der Datei
"test.txt" aus
print(listdir(".")) # gibt eine Liste der Dateien im aktuellen
Verzeichnis aus

# Wenn wir ein Paket mit einem anderen Namen importieren möchten,
können wir es mit dem Schlüsselwort "as" umbenennen
import os as o
print(o.getcwd()) # gibt das aktuelle Arbeitsverzeichnis aus
```

Dateien lesen und schreiben

Wie man Dateien in Python öffnet, liest und schreibt.

```
# Dateien lesen und schreiben

# Um eine Datei in Python zu öffnen, verwenden wir die Funktion open()
# Die Funktion benötigt den Dateinamen und den Modus, in dem die Datei
geöffnet werden soll
# Der Modus kann "r" für Lesen, "w" für Schreiben oder "a" für
Anhängen sein
# Wenn der Modus nicht angegeben wird, wird standardmäßig "r"
verwendet
```

```
# Hier ist ein Beispiel für das Öffnen einer Datei zum Lesen
datei = open("test.txt", "r")

# Wir können nun auf den Inhalt der Datei zugreifen, indem wir die
Methode read() verwenden
inhalt = datei.read()
print(inhalt)

# Wir sollten die Datei immer schließen, wenn wir fertig sind
datei.close()

# Hier ist ein Beispiel für das Öffnen einer Datei zum Schreiben
datei = open("neue_datei.txt", "w")

# Wir können nun in die Datei schreiben, indem wir die Methode write()
verwenden
datei.write("Dies ist ein Beispieltext.")

# Wir sollten die Datei immer schließen, wenn wir fertig sind
datei.close()

# Hier ist ein Beispiel für das Öffnen einer Datei zum Anhängen
datei = open("bestehende_datei.txt", "a")

# Wir können nun in die Datei schreiben, indem wir die Methode write()
verwenden
datei.write("Dies ist ein weiterer Beispieltext.")

# Wir sollten die Datei immer schließen, wenn wir fertig sind
datei.close()
```

Taschenrechner

Jetzt packen wir alles zusammen und erstellen einen einfachen Taschenrechner

```
# Taschenrechner App

# Zuerst definieren wir eine Funktion, die zwei Zahlen addiert
def addieren(x, y):
    return x + y

# Dann definieren wir eine Funktion, die zwei Zahlen subtrahiert
def subtrahieren(x, y):
    return x - y
```

```python
# Dann definieren wir eine Funktion, die zwei Zahlen multipliziert
def multiplizieren(x, y):
    return x * y

# Dann definieren wir eine Funktion, die zwei Zahlen dividiert
def dividieren(x, y):
    if y == 0:
        return "Division durch Null nicht möglich"
    else:
        return x / y

# Nun fragen wir den Benutzer nach der gewünschten Operation und den beiden Zahlen
print("Welche Operation möchtest du durchführen?")
print("1. Addieren")
print("2. Subtrahieren")
print("3. Multiplizieren")
print("4. Dividieren")

# Wir lesen die Eingabe des Benutzers ein und konvertieren sie in eine Ganzzahl
wahl = int(input("Deine Wahl (1/2/3/4): "))

# Wir lesen die Eingabe des Benutzers für die beiden Zahlen ein und konvertieren sie in Gleitkommazahlen
zahl1 = float(input("Erste Zahl: "))
zahl2 = float(input("Zweite Zahl: "))

# Wir führen die gewünschte Operation aus, indem wir die entsprechende Funktion aufrufen
if wahl == 1:
    print(zahl1, "+", zahl2, "=", addieren(zahl1, zahl2))

elif wahl == 2:
    print(zahl1, "-", zahl2, "=", subtrahieren(zahl1, zahl2))

elif wahl == 3:
    print(zahl1, "*", zahl2, "=", multiplizieren(zahl1, zahl2))

elif wahl == 4:
    print(zahl1, "/", zahl2, "=", dividieren(zahl1, zahl2))

else:
    print("Ungültige Eingabe")
```

Vielen Dank fürs lesen