Lernnachweis zu Kompetenz B2G:

**Kompetenz**:

B2G: Ich kann Funktionen als Objekte behandeln und diese in Variablen speichern und weitergeben.

**Lernnachweis**:

Während des Moduls 323 habe ich mich eingehend mit der Fähigkeit befasst, Funktionen als Objekte zu behandeln. Hier werde ich diese Kompetenz anhand eines Beispiels demonstrieren, bei dem Funktionen in Variablen gespeichert und weitergegeben werden.

**Beispiel: Funktionen als Objekte**

In Python können Funktionen wie jede andere Variable behandelt werden. Hier ist ein einfaches Beispiel, bei dem eine Funktion in einer Variable gespeichert und an eine andere Funktion übergeben wird:

**def** **gruss**(name):

**return** f"Hallo, {name}!"

**def** **rufe\_funktion\_auf**(func, argument):

**return** func(argument)

# Funktion in Variable speichern

meine\_funktion = gruss

# Funktion weitergeben und aufrufen

ergebnis = rufe\_funktion\_auf(meine\_funktion, "John")

**print**(ergebnis)

**Für Flask Applikation**:

**http://localhost:5000/begruessung mit den Parametern art (z.B., gruss oder abschied) und name (z.B., John) und du solltest die entsprechende Begrüßung erhalten**.

**Wichtig**: Curl oder Postman und sende eine POST-Anfrage.

**Lernprozess und Reflexion**:

Das Beherrschen der Funktionen als Objekte eröffnet neue Möglichkeiten für Flexibilität und Modularität im Code. Es ermöglicht, Funktionen dynamisch auszuwählen und zu kombinieren, was besonders nützlich ist, wenn Funktionen als Parameter an andere Funktionen übergeben werden müssen.

**Fazit**:

In meinem Lernprozess habe ich erkannt, dass diese Technik in der Funktionalen Programmierung besonders mächtig ist. Sie ermöglicht die Erstellung von Higher-Order-Funktionen, die Funktionen als Argumente akzeptieren oder Funktionen zurückgeben.

**Info**: Vollständiger Code im Flask Projekt (Github Repo).

Lernnachweis zu Kompetenz B2F:

**Kompetenz**:

B2F: Ich kann Funktionen als Argumente für andere Funktionen verwenden und dadurch höherwertige Funktionen erstellen.

**Lernnachweis**:

Während meines Studiums im Modul 323 habe ich mich intensiv mit der Fähigkeit auseinandergesetzt, Funktionen als Argumente für andere Funktionen zu verwenden. Hier werde ich diese Kompetenz anhand eines Beispiels demonstrieren, bei dem höherwertige Funktionen erstellt werden.

**Beispiel: Funktionen als Argumente**

In Python können Funktionen als Argumente für andere Funktionen übergeben werden. Hier ist ein einfaches Beispiel, bei dem eine Funktion übergeben wird, um eine Liste von Zahlen zu transformieren:

**def** **quadriere**(zahl):

**return** zahl \*\* **2**

**def** **kubiere**(zahl):

**return** zahl \*\* **3**

**def** **transformiere\_liste**(funktion, liste):

**return** [funktion(zahl) **for** zahl **in** liste]

zahlen = [**1**, **2**, **3**, **4**, **5**]

quadrate = transformiere\_liste(quadriere, zahlen)

kubiken = transformiere\_liste(kubiere, zahlen)

**print**("Quadrate:", quadrate)

**print**("Kubiken:", kubiken)

**Für Flask Applikation**: **http://localhost:5000/transformiere mit den Parametern operation (z.B., quadriere oder kubiere) und zahlen (z.B., 1,2,3,4,5)**

**Wichtig**: Curl oder Postman und sende eine POST-Anfrage.

**Lernprozess und Reflexion**:

Das Verwenden von Funktionen als Argumente ermöglicht die Schaffung von abstrakten Funktionen, die auf verschiedene Weisen angepasst werden können. Diese Technik fördert die Wiederverwendbarkeit von Code und ermöglicht die Erstellung allgemeinerer Funktionen.

**Fazit**:

Während meines Lernprozesses habe ich festgestellt, dass höherwertige Funktionen besonders nützlich sind, wenn wir bestimmte Muster oder Operationen auf verschiedene Arten anwenden möchten. Dies fördert einen modularen und flexiblen Codeansatz.

**Info**: Vollständiger Code im Flask Projekt (Github Repo).

Lernnachweis zu Kompetenz B2E:

**Kompetenz**:

B2E: Ich kann Funktionen als Objekte und Argumente verwenden, um komplexe Aufgaben. (Anwenden von Closures).

**Lernnachweis**:

Während meines Studiums im Modul 323 habe ich mich intensiv mit der Fähigkeit auseinandergesetzt, Funktionen als Objekte und Argumente zu verwenden, insbesondere unter Verwendung von Closures. Hier werde ich diese Kompetenz anhand eines Beispiels demonstrieren, bei dem Closures eingesetzt werden, um komplexe Aufgaben zu lösen.

**Beispiel: Verwendung von Closures**

Closures ermöglichen den Zugriff auf äußere Funktionen und ihre Variablen innerhalb einer inneren Funktion. Hier ist ein einfaches Beispiel mit einer Closure, die eine Zählfunktion implementiert:

**def** **erstelle\_zaehler**():

zahler = **0**

**def** **erhoehe\_zaehler**():

nonlocal zahler

zahler += **1**

**return** zahler

**return** erhoehe\_zaehler

# Erstelle einen Zähler

zaehler1 = erstelle\_zaehler()

**print**(zaehler1()) # Ausgabe: 1

**print**(zaehler1()) # Ausgabe: 2

**Für Flask Applikation**:

**http://localhost:5000/erhoehe\_zaehler**

**Lernprozess und Reflexion**:

Die Verwendung von Closures eröffnet Möglichkeiten für fortgeschrittenere und flexible Anwendungsfälle. Sie ermöglichen es, den Zustand innerhalb einer Funktion zu speichern und auf diesen Zustand zuzugreifen, ohne globale Variablen zu verwenden.

**Fazit**:

Während meines Lernprozesses habe ich festgestellt, dass Closures besonders nützlich sind, wenn wir eine Funktion benötigen, die einen bestimmten Zustand behält und diesen bei jedem Aufruf aktualisiert. Sie sind eine leistungsstarke Technik, um komplexe Aufgaben in einer klar strukturierten und modularen Weise zu lösen

**Info**: Vollständiger Code im Flask Projekt (Github Repo).