Lernnachweis zu Kompetenz B2F

Verwendung von Funktionen als Argumente für höherwertige Funktionen

Einleitung

Die Kompetenz B2F bezieht sich darauf, Funktionen als Argumente für andere Funktionen zu verwenden, um dadurch höherwertige Funktionen zu erstellen. Dieser Ansatz, bekannt als "Funktionen höherer Ordnung" (Higher-Order Functions), ermöglicht eine elegante und flexible Programmierung. In dieser Dokumentation werden wir den Einsatz von Funktionen als Argumente anhand von Python-Beispielen erläutern.

Warum ist die Verwendung von Funktionen als Argumenten wichtig?

Die Verwendung von Funktionen als Argumenten erlaubt die dynamische Anpassung des Verhaltens von Funktionen. Dies fördert die Wiederverwendbarkeit von Code und ermöglicht es, allgemeine Funktionen zu schreiben, die für verschiedene Anwendungsfälle angepasst werden können.

Beispiel: Funktion als Argument für höherwertige Funktion

```
def apply_operation(func, x, y):
    return func(x, y)

def add(x, y):
    return x + y

def multiply(x, y):
    return x * y

# Funktionen als Argumente übergeben
result_add = apply_operation(add, 3, 4)
print(result_add)  # Ausgabe: 7

result_multiply = apply_operation(multiply, 3, 4)
print(result_multiply)  # Ausgabe: 12
```

In diesem Beispiel wird die Funktion «apply_operation» erstellt, die eine Operation auf zwei Zahlen anwendet. Die Funktionen «add» und «multiply» werden als Argumente übergeben, was es ermöglicht, verschiedene Operationen mit derselben höherwertigen Funktion durchzuführen.

Beispiel: Filtern von Elementen mit einer höherwertigen Funktion

```
def filter_elements(predicate, elements):
    return [elem for elem in elements if predicate(elem)]

def is_even(x):
    return x % 2 == 0

def is_positive(x):
    return x > 0

numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

even_numbers = filter_elements(is_even, numbers)
print(even_numbers)  # Ausgabe: [2, 4, 6, 8, 10]

positive_numbers = filter_elements(is_positive, numbers)
print(positive_numbers)  # Ausgabe: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Hier wird die Funktion «filter_elements» erstellt, die eine höherwertige Funktion «predicate» verwendet, um Elemente in einer Liste zu filtern. Die Funktionen «is_even» und «is_positive» werden als Argumente übergeben, um verschiedene Filteroperationen durchzuführen.

Fazit

Die Verwendung von Funktionen als Argumente ermöglicht die Erstellung von höherwertigen Funktionen, die flexibel und anpassbar sind. Dieser Ansatz fördert eine elegante und modularisierte Programmierung, indem allgemeine Funktionen geschrieben werden, die für verschiedene Kontexte verwendet werden können. Die Beispiele verdeutlichen die praktische Anwendung dieser Kompetenz in der Programmierung.