Lernnachweis zu Kompetenz A1G

> 💡 Kompetenz: Ich kann die Eigenschaften von Funktionen beschreiben (z.B. pure function) und den Unterschied zu anderen Programmier-Strukturen erläutern (z.B. zu Prozedur).

✅ Lernnachweis: Beschreiben der Eigenschaften von Funktionen

In diesem Lernfeld habe ich mich mit den Eigenschaften von Funktionen und deren Unterschied zu anderen Programmierstrukturen befasst. Was für mich eher schwierig war, war die verschiedenen Programmierstrukturen zu verstehen, welche ich zuvor noch nicht kannte oder von denen ich nicht wusste, dass die ein eigenes Ding sind, ein Beispiel dazu wären Prozeduren, etwas, was ich schon oft benutzt habe aber vom Namen allein nicht gewusst hätte, dass diese damit gemeint sind.

⚡ Funktionen:

In der Programmierung sind Funktionen etwas, was dem Programmierer erlaubt einzelne Teile des Codes immer wieder zu nutzen.

Bei den **Pure Functions** ist die Besonderheit, dass sie immer wieder das gleiche Resultat liefern, sofern sie denselben Input erhalten.

Beispiel aus dem Projekt:

```
def pure_function(value):
    return str(value)
```

Impure Functions jedoch können beim gleichen Input immer wieder verschiedene Resultate liefern

Beispiel aus dem Projekt:

```
from random import random

def impure_function(value):
    return str(random() * 16 * value)
```

Der Unterschied zu **Prozeduren** beispielsweise liegt darin, dass eine solche globale Zustände verändern kann, während Funktionen dies nicht dürfen. Eine Prozedur kann auch einfach ein Ablauf von Anweisungen sein und muss nicht dringenderweise einen Wert zurücksenden.

Beispiel:

```
def update_counter():
    global counter
    counter += 1
```

🧠 Lernprozess:

Ich hatte nicht wirklich allzu grosse Schwierigkeiten bei dem Erlernen dieser Kompetenz. Ich konnte jedoch davon Profitieren zu erlernen was die verschiedenen Arten von Programmierstrukturen sind, da ich diese jetzt je nach Anwendungsfall besser verwenden kann.

🕐 Zukünftige Schritte:

Ich werde versuchen in der weiteren Zukunft mehr darauf zu achten welche Strukturen ich verwenden und meinen Code so anpassen, dass er sich mehr an einer einzelnen Struktur haltet, um so einheitlicheren Code zu schreiben.

Lernnachweis zu Kompetenz A1F

> 💡 Kompetenz: Ich kann das Konzept von *immutable values* erläutern und dazu Beispiele anwenden. Somit kann ich dieses Konzept funktionaler Programmierung im Unterschied zu anderen Programmiersprachen erklären (z.B. im Vergleich zu referenzierten Objekten)

✅ Lernnachweis: Das Konzept von immutable values erklären

In diesem Lernfeld habe ich mich mit immutable values und die unterschiede in der funktionalen Programmierung im unterschied zu anderen Programmiersprachen befasst. Ebenfalls habe ich über referenzierte Objekte informiert, etwas, von dem ich zuvor gar nicht wusste wie es funktioniert.

⚡ Funktionen:

Immutable values sind Werte, welche nicht verändert werden können/dürfen ein Beispiel in Python dafür wäre so etwas wie Tuples. Um diese zu verändern muss man ein neues erstellen, mit den Änderungen die man haben will. Das Gegenteil wäre zum Beispiel eine Liste, diese können beliebig verändert werden. Man kann sich jedoch trotzdem an das Konzept der Immutabilität halten indem man Objekte einfach nicht verändert sondern einfach neue mit den Änderungen erstellt.

Beispiel aus dem Projekt:

```
def immutable_function(new_list):
    if not new_list:
        return 0
    for i in new_list:
        updated = new_list[1:]
        return i + immutable_function(updated)

result = immutable_function([1, 2, 3, 4, 5])
return str(result)
```

Man muss jedoch die Referenzierung der Objekte nicht vernachlässigen. Bei sprachen wie Python und Java werden zum Beispiel werden Objekte nicht in einfachen Variablen gespeichert sondern als eine Referenz auf deren Speicheradresse daher kann es passieren, dass man eine List in Python in einer anderen Variable als Kopie speichern möchte, aber schlussendlich beim abändern der einen auch die andere bearbeitet.

Beispiel:

```
# Erstellen eines referenzierten Objekts (in diesem Fall eine Liste)
original_list = [1, 2, 3]

# Referenzieren des Objekts durch eine andere Variable statt Kopieren
referenced_list = original_list

# Änderung des referenzierten Objekts über die zweite Variable
referenced_list.append(4)

# Ausgabe der Veränderungen in beiden Variablen
print("Original List:", original_list)
print("Referenced List:", referenced_list)

# Ausgabe:
>>> Original List: [1, 2, 3, 4]
>>> Referenced List: [1, 2, 3, 4]
```

🧠 Lernprozess:

Ich hatte schon einmal den Fehler in einem anderen Programm, wo ich ein Objekt bearbeitet hatte aber sich auf einmal ein anderes Objekt verändert hat. Ich bin dann auf StackOverfow gegangen und habe dort nach Antworten gesucht. Ich stiess dann auch auf den Begriff “Referenzierung”, hatte mir aber damals keine Mühe gemacht den Begriff genauer unter die Lupe zu nehmen. Jetzt weiss ich, wie ich dies zu vermeiden habe und mir ist bewusst was gemacht werden kann als Alternative.

Ich weiss auch was ich beachten muss, wenn ich mit Immutabilität arbeite.

🕐 Zukünftige Schritte:

In der Zukunft werde ich definitiv weniger Probleme mit referenzierten Objekten haben, da ich ja jetzt weiss wie das funktioniert, auch werde ich versuchen, wenn ich objektorientiert programmiere, mich an das Konzept der Immutabilität zu halten, auch wenn die Objekte nicht direkt über eine Absicherung gegen Mutation verfügen.

Endpoint für die Flask Applikation: (localhost:5000)/A1F

Lernnachweis zu Kompetenz A1E

> 💡 Kompetenz: Ich kann aufzeigen wie Probleme in den verschiedenen Konzepten (OO, prozedural und funktional) gelöst werden und diese miteinander vergleichen.

✅ Lernnachweis: Probleme mit verschiedenen Konzepten Lösen/Vergleichen

Hier habe ich mich mit den verschiedenen Konzepten der Programmierung auseinandergesetzt, wobei ich mich hauptsächlich auf Python Objektorientiertes Programmieren (POOP), Funktionales Programmieren und Prozedurales Programmieren fokussiert habe. Diese Programmierungskonzepte habe ich miteinander verglichen und recherchiert was wann angewendet wird.

⚡ Funktionen:

Die Idee vom **objektorientierten Programmieren** ist, den Code in verschiedene Klassen zu unterteilen, in welchen die verschiedenen Funktionen abgerufen werden können.

Beispiel aus dem Projekt:

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        return f"Brand: {self.brand}, Model: {self.model}"

my_car = Car("Toyota", "Corolla")
return my_car.display_info()
```

Bei **Funktionalem Programmieren** konzentriert man sich mehr auf individuelle Funktionen als Bausteine eines Codes und man arbeitet mit Konzepten wie zum Beispiel Immutabilität.

Beispiel aus dem Projekt:

```
def multiply_numbers(a, b):
    return a * b

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

return f"3 and 5:<br><br>added = {add_numbers(3, 5)}<br><br>multiplied = {multiply_numbers(3, 5)}"
```

Und zum Schluss geht es bei **prozeduraler Programmierung** darum sich mehr auf die Anweisungen und Verwendungen von Funktionen im Code zu fokussieren, bzw. es geht eher darum einen Ablauf zu machen der einfach an das erwünsche Ziel kommt.

Beispiel aus dem Projekt:

```
def add_and_multiply(val):
    added = val + 3
    multiplied = val * 3
    return f"3 and 10... added: {added}, multiplied: {multiplied}"

value = 10
return str(add_and_multiply(value))
```

🧠 Lernprozess:

Zwar hatte ich schon dank einem Vorherigen Modul schon Erfahrung mit dem Konzept des OOP, jedoch musste ich zuerst nochmals ein wenig auffrischen, wie das ganze funktioniert, da ich das auch seit längerem nicht mehr gemacht hatte. Normalerweise arbeite ich nämlich eher in einem prozeduralen Stil, da es für mich persönlich am einfachsten ist. Einige der Grundkonzepte der funktionalen Programmierung waren für mich Anfangs auch eher schwierig zu verstehen, jedoch ging es nicht sehr lange bis ich die auch im Griff hatte.

🕐 Zukünftige Schritte:

Im Praktikum werde ich hauptsächlich mit Java arbeiten, was eine Programmiersprache ist in der man praktisch nur mit OOP arbeitet, weswegen ich froh war, dass ich es nochmals ein wenig wiederholen konnte. Ich werde aber wahrscheinlich wenn ich an meinen eigenen Projekten arbeite versuchen mich von Prozeduralen abzuwenden und mehr auf das

Funktionale gehen, wenn es mit OOP nicht sehr gut lösbar ist oder es einfach keinen Sinn mach so zu arbeiten.

Endpoint für die Flask Applikation: (localhost:5000)/A1E

Lernnachweis zu Kompetenz B1G

> 💡 Kompetenz: Ich kann ein Algorithmus erklären.

✅ Lernnachweis: Einen Algorithmus erklären

Ich habe mich in diesem Lernfeld mit Algorithmen und was sie eigentlich sind auseinandergesetzt. Was mir hierbei ein wenig schwergefallen ist, war es bei den Aufgaben zu sehen, wie ich die Algorithmen eigentlich implementieren sollte. Jedoch bin ich schnell vorwärtsgekommen, sobald ich eine Idee hatte oder mir ein Anfang eines möglichen Lösungsansatzes eingefallen ist.

⚡ Funktionen:

Im Grunde ist ein Algorithmus eine Ansammlung von genauen Anweisungen zur Lösung eines Problems, welche dann vom Computer ausgeführt werden. Der Unterschied zu den Funktionen oder anderen Prozessen liegt darin, dass der Algorithmus auf ein spezifisches Problem fokussiert ist.

Beispiel aus dem Projekt:

```
def einfacher_algorithmus(zahl):
    ergebnis = zahl * 3
    return ergebnis

ergebnis_einfach = einfacher_algorithmus(5)
return ergebnis_einfach
```

🧠 Lernprozess:

Anfangs war es für mich ein wenig schwierig die Ideen hinter den Algorithmen zu erkennen, da sie meiner Meinung nach einen speziellen Denkprozess erfordern, welchen ich noch nicht im Kopf hatte. Beispielsweise verwirrte mich zunächst der Algorithmus des Turm von Hanoi, jedoch konnte ich mich dann aber an die Denkweise anpassen und es funktionierte schlussendlich auch meine eigenen Algorithmen zu entwickeln.

🕐 Zukünftige Schritte:

In der Zukunft werde ich wahrscheinlich nicht sehr viel mit Algorithmen zu tun haben, jedoch finde ich das Wissen sehr nützlich, da es beim Problemlösen eine grosse Hilfe sein kann einen Algorithmischen Denkprozess zu verwenden. Zudem habe ich noch ein kleinere Projekt zu Robotik am laufen, bei dem Algorithmen von grosser Hilfe sind. Dabei handelt es sich um eine eigene, kleinere Version des Wettbewerbs “Micromouse”, welches ich zum üben gerne programmieren würde.

Endpoint für die Flask Applikation: (localhost:5000)/B1G

Lernnachweis zu Kompetenz B1F

> 💡 Kompetenz: Ich kann Algorithmen in funktionale Teilstücke aufteilen.

✅ Lernnachweis: Algorithmen in funktionale Teilstücke aufteilen

In diesem Kompetenzfeld habe ich mich mit der Aufteilung von Algorithmen in ihre funktionalen Teilstücke befasst. Das war Anfangs zwar ein wenig kompliziert, jedoch hatte ich es dann auch schnell im Griff.

⚡ Funktionen:

Die Aufteilung von Algorithmen in funktionale Teilstücke ist eigentlich gar nicht so schwierig, sobald man versteht um was es geht. Die Idee dahinter ist es die Funktionen und Algorithmen zu vereinen und einzelne Teilen eines Algorithmus in Funktionen zu verlagern und diese dann im grösseren Gesamtalgorithmus zu vereinigen. Das mach, dass der Algorithmus einfacher zu verstehen, warten und wiederverwenden ist.

Beispiel us dem Projekt:

```
def multipliziere_mit_drei(zahl):
    return zahl * 3

def fuege_zusammen(ergebnis):
    ergebnis = f"ergebnis: {ergebnis}"
    return ergebnis

ergebnis_funktional = fuege_zusammen(multipliziere_mit_drei(5))
return ergebnis_funktional
```

🧠 Lernprozess:

Es war nicht wirklich schwierig für mich die Idee hinter der funktionalen Aufteilung der Funktionen zu sehen und dann auch in den Aufgaben anwenden zu können, daher habe ich nicht allzu viel neues dazugelernt. Dennoch finde ich es wichtig es sich noch einmal genauer anzusehen.

🕐 Zukünftige Schritte:

Sollte ich in der Zukunft jemals in der Arbeit oder Privat mit Algorithmen arbeiten werde ich sie höchstwahrscheinlich in funktionale Teilstücke aufteilen, da es einfach viel verständlicher ist, als einfach alles so zu lassen, wie es halt geschrieben wurde und, weil es dadurch auch für Andere verständlicher ist, sollten Andere damit arbeiten wollen.

Endpoint für die Flask Applikation: (localhost:5000)/B1F

Lernnachweis zu Kompetenz B1E

> 💡 Kompetenz: Ich kann Funktionen in zusammenhängende Algorithmen implementieren.

✅ Lernnachweis: Funktionen in Algorithmen implementieren

Hier habe ich mich mit der Implementierung von Funktionen in zusammenhängende Algorithmen auseinandergesetzt, was mich zunächst verwirrte, da ich dachte es sei so ziemlich das selbe wie bei Kompetenz B1F, jedoch wurde mir später bewusst, dass es kleine Unterschiede zwischen den beiden Bezeichnungen gibt.

⚡ Funktionen:

Der Unterschied zur vorherigen Kompetenz liegt eigentlich darin, dass es sich hier viel mehr darum dreht die Funktionen in den Algorithmus zu implementieren und sie in einer spezifischen Reihenfolge ablaufen zu lassen. Also das heisst, es geht darum alles in einer klaren Koordination von Sequenzen aus Funktionen ablaufen zu lassen.

Beispiel aus dem Projekt:

```
def multipliziere_mit_drei(zahl):
    return zahl * 3

def fuege_zusammen(ergebnis):
    ergebnis = f"ergebnis: {ergebnis}"
    return ergebnis

def zusammenhaengender_algorithmus(zahl):
    ergebnis = multipliziere_mit_drei(zahl)
    ergebnis = fuege_zusammen(ergebnis)

    ergebnis *= 2

    return ergebnis
```

```
ergebnis_zusammenhaengend = zusammenhaengender_algorithmus(5)
print(ergebnis_zusammenhaengend)
```

🧠 Lernprozess:

Ich war wie bereits gesagt am Anfang ein wenig verwirrt, da ich dachte diese Kompetenz sei das gleiche wie die vorherige, nachdem ich mich aber genauer damit auseinandergesetzt hatte habe ich realisiert, dass es doch den Unterschied im Fokus/der Philosophie des ganzen gibt.

🕐 Zukünftige Schritte:

Meiner Meinung nach ist der Unterschied zwischen der Kompetenz vor dieser und dieser hier so klein, dass es sich eigentlich um dasselbe handelt, es sei denn ich bin einfach verwirrt und habe die Kompetenz falsch verstanden, was ich aber nicht vermute. Daher heisst es auch hier, sollte ich je Algorithmen für ein Projekt verwenden, dann werde ich es in Funktionen aufteilen um es für alle Beteiligten zu vereinfachen.

Endpoint für die Flask Applikation: (localhost:5000)/B1E

Lernnachweis zu Kompetenz B2G

> 💡 Kompetenz: Ich kann Funktionen als Objekte behandeln und diese in Variablen speichern und weitergeben.

✅ Lernnachweis: Funktionen als Objekte behandeln und in Variablen speichern

In diesem Lernnachweis habe ich mich über die Verwendung von Funktionen als Objekte und deren Anwendung in Verbindung mit Variablen schlau gemacht. zunächst war dies zwar ein wenig kompliziert zu verstehen, jedoch ist mir mittlerweile bewusst wie ich das anzugehen habe und ich kann dies Korrekt in meinem eigenen Code anwenden.

⚡ Funktionen:

Beim Recherche betreiben habe ich herausgefunden, dass in vielen Programmiersprachen, darunter auch Python, Funktionen als sogenannte "First-Class Citizens" behandelt werden. Das bedeutet, dass Funktionen wie jede andere Datenart behandelt werden können, einschließlich der Möglichkeit, sie in Variablen zu speichern und an andere Funktionen weiterzugeben. Etwas, was mir zuvor noch nicht wirklich bewusst war, obwohl ich es bereits einmal gesehen hatte.

Beispiel aus dem Projekt:

```
def greet_person(name):
    return f"Hello, {name}!"

my_greeting = greet_person

return my_greeting("Visitor")
```

🧠 Lernprozess:

Am Anfang wusste ich noch nicht sehr genau was mit dem einspeichern von Funktionen als Variable gemeint war, jedoch wurde mir sehr schnell klar, dass es eigentlich nichts zu schwieriges ist. Grundsätzlich ist es sogar ein einfaches umbenennen einer Funktion. Dies hatte ich zuvor auch schon angewendet aber nicht wirklich als solch eine Interaktion in Erinnerung behalten.

🕐 Zukünftige Schritte:

In der Zukunft bezweifle ich, dass ich wirklich viel damit arbeiten werde, allerdings kann es natürlich immer sein, dass es Mal sein muss, auf diese Art zu arbeiten, und da ist es natürlich immer besser das Wissen bereits zu besitzen.

Endpoint für die Flask Applikation: (localhost:5000)/B2G

Lernnachweis zu Kompetenz B2F

> 💡 Kompetenz: Ich kann Funktionen als Argumente für andere Funktionen verwenden und dadurch höherwertige Funktionen erstellen.

✅ Lernnachweis: Funktionen als Argumente für andere Funktionen verwenden.

Dieses Kompetenzfeld war meine Einführung zu sogenannten “higher order functions” oder auf Deutsch “höherwertige Funktionen”. Diese waren zwar anfangs ein wenig schwieriger zu verstehen, im Sinne von weswegen das funktionieren sollte, nicht im Sinne von der Programmieren, jedoch ist das schon längst kein Problem mehr und ich bin jetzt in der Lage alle Arten von higher order functions zu erstellen.

⚡ Funktionen:

Im Grunde geht es hier einfach darum, dass man einer Funktion einfach eine andere Funktion als Argument mitgeben kann, welche dann in der Ersten drin ausgeführt werden kann. Das tönt zunächst ein wenig verwirrend, wird jedoch mit einem Beispiel schnell verständlich.

Beispiel aus dem Projekt:

```
def higher_order(function_name, value1, value2):
    return function_name(value1, value2)

def add_values(val1, val2):
    return val1 + val2

def subtract_values(val1, val2):
    return val1 - val2

return f"5 and 8: added: {higher_order(add_values, 5, 8)}, " \
       f"subtracted: {higher_order(subtract_values, 5, 8)}"
```

🧠 Lernprozess:

Es war Anfangs ein wenig verwirrend, wieso das diese Funktionen noch funktionieren, jedoch ist das Programmieren von Anfang an nicht wirklich ein Problem gewesen, da ich mir nur einige Beispiele ansehen musste und dann alles klar war. Zwar weiss ich immer noch nicht wieso es funktioniert, jedoch weiss ich, dass es funktioniert und wie ich es zum funktionieren bringe, was schlussendlich auch sehr wichtig ist.

🕐 Zukünftige Schritte:

Ich bin mir nicht wirklich sicher ob ich dies in der Zukunft noch einmal antreffen werde, jedoch ist es sehr wohl möglich, da es auch in Java existiert, was die Sprache sein wird in der ich im Praktikum arbeiten muss.

Endpoint für die Flask Applikation: (localhost:5000)/B2F

Lernnachweis zu Kompetenz B2E

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

✅ Lernnachweis: Anwenden von Closures

Closures sind etwas, was mich zwar auch noch ein wenig verwirrt, jedoch weiss ich mittlerweile wie ich sie anwenden kann, ohne zu viel wissen zu müssen. Die Schwierigkeit liegt darin, dass es nicht wirklich eine gewöhnliche Art zu Programmieren ist, jedoch sind Closures mächtige Konstrukte, die einen Code um ein vielfaches komplexer machen können.

⚡ Funktionen:

Wenn man sich Closures anschaut kann man es sich eigentlich so vorstellen, dass man ein Funktion in einer Funktion hat. Die innere Funktion dient dabei aber als eine kleiner Speicher der sich an einen oder mehr Werte “erinnert”, welche die äussere Funktion benötigt, wenn sie ausgeführt wird.

Beispiel aus dem Projekt:

```
def multiplication(factor):
    def multiply(num):
        return num * factor

    return multiply

double_value = multiplication(2)
tripple_value = multiplication(3)

return f"Value 5: doubled: {double_value(5)}, tripled: {tripple_value(5)}"
```

🧠 Lernprozess:

Zuerst hatte ich von den Closures absolut nichts verstanden, mir war einfach nicht klar, wie es möglich sein sollte, dass die Funktion sich dann an den ersten Wert erinnert der innerdrinn gespeichert wird. Dann habe ich aber online unter verschiedenen Quellen nachgeschaut wie das ganze eigentlich Funktioniert und irgendwann hat es dann Sinn gemacht. Zwar ist die genaue Funktionsweise noch ein wenig schwierig, aber ich weiss trotzdem wie ich das anwenden kann.

🕐 Zukünftige Schritte:

In der Zukunft, kann es durchaus sein, dass ich mit diesen Funktionen zu tun haben werde, jedoch bin ich mir sicher, dass es dann ein wenig anders aussehen würde, da es dann ja in Java und nicht in Python wäre. deswegen müsste ich es mir so oder so no einmal anschauen. Das Grundwissen, was ich bei diesem Kompetenzkasten erhalten habe wird mir sicher eine Hilfe sein.

Endpoint für die Flask Applikation: (localhost:5000)/B2E

Lernnachweis zu Kompetenz B3G

> 💡 Kompetenz: Ich kann einfache Lambda-Ausdrücke schreiben, die eine einzelne Operation durchführen, z.B. das Quadrieren einer Zahl oder das Konvertieren eines Strings in Großbuchstaben.

✅ Lernnachweis: Einfache Lambda-Ausdrücke schreiben

Ich habe mich in diesem Kompetenzfeld mit der Erstellung einfacher Lambda-Ausdrücke befasst. Diese sind zwar Anfangs möglicherweise ein wenig schwierig zu verstehen, jedoch wird es um einiges einfacher, sobald man einmal versteht wie es Funktioniert.

⚡ Funktionen:

Einfache Lambda-Ausdrücke sind eine einfache Art Funktionen zu schreiben die man nicht zuerst definieren muss (Anonyme Funktionen). Sie sind sehr viel kompakter und schneller zu implementieren als für ein kleines Problem eine ganze Funktion zu bauen.

Beispiel aus dem Projekt:

```
square_root = lambda x: x ** 0.5
    return f"The square root of 56: {square_root(56)}"
```

🧠 Lernprozess:

Ich vermute in Zukunft vermehrt Lambda-Ausdrücke zu verwende, da sie in vielen Fällen viel einfacher und schneller zu implementieren sind als ganze Funktionen für simple Probleme oder Berechnungen. Auch weil es diese in Java gibt bin ich mir sicher, dass ich diese noch antreffen werde.

🕐 Zukünftige Schritte:

Ich vermute in Zukunft vermehrt Lambda-Ausdrücke zu verwende, da sie in vielen Fällen viel einfacher und schneller zu implementieren sind als ganze Funktionen für simple Probleme oder Berechnungen. Auch weil es diese in Java gibt bin ich mir sicher, dass ich diese noch antreffen werde.

Endpoint für die Flask Applikation: (localhost:5000)/B3G

Lernnachweis zu Kompetenz B3F

> 💡 Kompetenz: Ich kann Lambda-Ausdrücke schreiben, die mehrere Argumente verarbeiten können.

✅ Lernnachweis: Lambda-Ausdrücke mit mehreren Argumenten

Hier habe ich mich wie im letzten Kompetenzfeld mit den Lambda-Ausdrücken auseinander gesetzt, mit dem Unterschied, dass sie dieses Mal mehr als nur ein Argument akzeptieren können. Dies ist aber nicht viel komplizierter als solche mit nur einem Argument.

⚡ Funktionen:

Der einzige Unterschied im Vergleich zu den einfachen Lambda-Ausdrücken ist die Anzahl Argumente die in der Lambda Funktion angegeben werden, sonst ist alles genau gleich.

Beispiel aus dem Projekt:

```
multiply = lambda a, b, c: a * b * c
return f"multipliziere 5, 7 und 12 zusammen: {multiply(5, 7, 12)}"
```

🧠 Lernprozess:

Wie bereits erwähnt hatte ich dank dem Script4Fun Freifach schon sehr viel Erfahrung mit Lambda, wobei ich jedoch anmerken sollte, dass es eigentlich an CodeWars lag und nicht dem eigentlichen Freifach. Daher war es immer noch sehr einfach wieder in das ganze einzusteigen und wieder zu wissen wie das Funktioniert.

🕐 Zukünftige Schritte:

Nochmals, wie bereits erwähnt bin ich mir sicher, dass ich nicht zum letzten mal mit Lambda in Kontakt getreten bin, sicher werde ich es mir in Java noch ein weiteres Mal genauer anschauen und ich werde sicher im Praktikum damit arbeiten müssen, da es einfach eine Praktische Lösung ist um nicht ganze Funktionen implementieren zu müssen.

Endpoint für die Flask Applikation: (localhost:5000)/B3F

Lernnachweis zu Kompetenz B3E

> 💡 Kompetenz: Ich kann Lambda-Ausdrücke verwenden, um den Programmfluss zu steuern, z.B. durch Sortieren von Listen basierend auf benutzerdefinierten Kriterien.

✅ Lernnachweis: Programmflüsse steuern mit Lambda-Ausdrücken

In diesem Lernfeld habe ich mich mit dem Steuern von Programflüssen mit Hilfe von Lambda-Ausdrücken auseinandergesetzt indem ich verschiedene Aufgaben wie das Sortieren von Listen oder ähnlichem befasst habe. Schwierigkeiten dabei waren je nach Aufgabe verschiedene Arten der Implementierung von Lambda zu verwenden.

⚡ Funktionen:

In Python werden Lambda Funktionen in diesem Fall der Programmflusssteuerung mit Hilfe von den Funktionen `map`, `filter`, `sorted` und `reduce` verwendet. Mit diesen können Lambda-Ausdrücke vorübergehend definiert werden, um dann in den zuvor genannten Funktionen den Programmfluss zu steuern.

Beispiel aus dem Projekt:

```
sort_by_length = lambda word: len(word)

list_of_words = ["short", "ThisIsAVeryLongWordWithManyLetters", "NormalWord", "WordButLonger", "E"]

sort_list = sorted(list_of_words, key=sort_by_length)

return f"sorted list: {sort_list}<br>original list: {list_of_words}"
```

🧠 Lernprozess:

Obwohl ich zuvor schon mit den Lambda-Ausdrücken gearbeitet hatte, waren mir diese Funktionen eigentlich noch eher neu, da ich diese noch nie oder fast nie verwendet hatte. Es war aber kein grosser Aufwand diese dazuzulernen, da es im Grunde ja nur einfache Lambda-Ausdrücke sind die dann als Key oder etwas Ähnliches verwendet werden.

🕐 Zukünftige Schritte:

In der Zukunft werde ich möglicherweise ein wenig mit dieser Art von Implementierung für Lambda-Ausdrücke arbeiten, da es natürlich einige dinge stark vereinfachen kann, jedoch kann ich mir auch gut vorstellen, dass es nicht nötig sein könnte oder einfach andere Lösungsansätze einfacher scheinen. Das liegt hauptsächlich daran, dass ich jetzt nicht sehr viel damit arbeite und es daher ungewohnt und ineffizient ist diese zu verwenden.

Endpoint für die Flask Applikation: (localhost:5000)/B3E

Lernnachweis zu Kompetenz B4G

> 💡 Kompetenz: Ich kann die Funktionen Map, Filter und Reduce einzeln auf Listen anwenden.

✅ Lernnachweis: Map, Filter und Reduce einzeln auf Listen anwenden

Hier habe ich mich mit der Verwendung von den Funktionen Map, Filter und Reduce zur Verarbeitung von Listen angeschaut. Eigentlich ist dies ziemlich simpel und ich habe schon ziemlich oft in Projekten und anderen Aufgaben damit gearbeitet, da man sich schnell durch Listen durcharbeiten kann.

⚡ Funktionen:

Die **Map Funktion** ist eine Higher Order function, welche als Argumente eine Funktion und eine oder mehrere itterables entgegennimmt, in diesem Fall eine Liste. Was es macht ist, bei jedem der Values im itterable die Funktion ausführt.

Beispiel aus dem Projekt:

```
numbers = [1, 2, 3, 4, 5, 6]
square_list = list(map(lambda x: x ** 2, numbers))
```

Die **Filter Funktion** funktioniert im Grunde genau gleich, was sich aber unterscheidet ist, dass hier die Funktion dazu da ist zu selektionieren welche Values bleiben können.

Beispiel aus dem Projekt:

```
numbers = [1, 2, 3, 4, 5, 6]
even_numbers = list(filter(lambda x: x % 2 == 0, numbers))
```

Zuletzt noch die **Reduce Funktion**, welche ebenfalls auf eine ähnliche Art funktioniert. Was sie macht ist, die Funktion auf alle Elemente im Itterable durchzuführen und dann die Ergebnisse zusammenführt.

Beispiel aus dem Projekt:

```
numbers = [1, 2, 3, 4, 5, 6]
sum_of_list = reduce(lambda x, y: x + y, numbers)
```

🧠 Lernprozess:

Vor diesem Modul hatte ich zumindest von Map und Reduce noch nie gebrauch gemacht, da ich nicht wusste, wie sie funktionieren. Nachdem ich mir jedoch einige Beispiele angeschaut hatte, war ich dann in der Lage die Aufgaben zu lösen und ich habe dann auch zuhause weiter nachgeschaut wie diese Funktionieren, sodass ich jetzt sehr gut weiss wie ich sie anwenden kann.

🕐 Zukünftige Schritte:

In der Zukunft werde ich wahrscheinlich schon noch mit solchen Funktionen zu tun haben, wobei ich jedoch noch nachschauen müsste wie genau das mit Java funktioniert, da ich ja im Praktikum nur mit dem Arbeiten werde.

Endpoint für die Flask Applikation: (localhost:5000)/B4G

Lernnachweis zu Kompetenz B4F

> 💡 Kompetenz: Ich kann Map, Filter und Reduce kombiniert verwenden, um Daten zu verarbeiten und zu manipulieren, die komplexere Transformationen erfordern.

✅ Lernnachweis: Map, Filter und Reduce kombinieren für Datenverarbeitung

Das Kombinieren von Map, Filter und Reduce Funktionen für komplexe Datenverarbeitung ist einfach solange man die Grundsätzlichen Ideen hinter diesen Funktionen versteht und was sie eigentlich genau machen.

⚡ Funktionen:

Wie gesagt ist es nicht sehr schwierig die Funktionen zu Kombinieren, wenn man versteht wie sie Funktionieren, was ich bereits im vorherigen Kompetenzfeld erklärt habe. Es muss jedoch beachtet werden, dass man schnell sehr unübersichtlichen Code schreiben kann, wenn man nicht aufpasst was man genau macht.

Beispiel aus dem Projekt:

```
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

result = reduce(lambda x, y: x + y, map(lambda x: x**2, filter(lambda x: x % 2 != 0, numbers)))

return f"List of numbers: {numbers}<br>Combined function (sum of the square of all odd numbers): {result}"
```

🧠 Lernprozess:

Ich hatte von Anfang an nicht wirklich grosse Schwierigkeiten bei diesem Thema, da ich es ja relativ schnell verstanden hatte. Allerding war dieser Teil ein wenig schwierig, da ich beim Programmieren des Projektes ein wenig unübersichtlich gearbeitet habe und es dann ein wenig kompliziert zu entziffern war, was jedoch hilft sind Kommentare und die Formatierung zu ändern.

🕐 Zukünftige Schritte:

Ich werde versuchen in der weiteren Zukunft mehr darauf zu achten welche Strukturen ich verwenden und meinen Code so anpassen, dass er sich mehr an einer einzelnen Struktur haltet, um

so einheitlicheren Code zu schreiben.

Endpoint für die Flask Applikation: (localhost:5000)/B4F

Lernnachweis zu Kompetenz B4E

> 💡 Kompetenz: Ich kann Map, Filter und Reduce verwenden, um komplexe Datenverarbeitungsaufgaben zu lösen, wie z.B. die Aggregation von Daten oder die Transformation von Datenstrukturen.

✅ Lernnachweis: Komplexe Datenverarbeitung mit Map, Filter und Reduce

Hier habe ich mit Hilfe der Map, Filter und Reduce Funktionen erlernt wie genau ich komplexere Datenverarbeitungsaufgaben lösen kann. Es war eigentlich von der Grundidee genau das gleiche wie das Vorherige Kompetenzfeld, einfach ein wenig schwieriger und mit komplexeren Daten.

⚡ Funktionen:

Wie bereits erwähnt war dieses Kompetenzfeld so ziemlich das selbe wie das vorherige, mit ein wenig komplizierteren Daten, was eigentlich nicht sehr viel anders verarbeitet werden musste.

Beispiel aus dem Projekt:

```
students = [
        ("Bucac", 18, 4.7),
        ("Bobin", 19, 5),
        ("AƔA", 18, 4.25),
        ("Ferrari", 18, 5.1),
        ("Nichtola", 17, 6.0)
    ]

    avg_grade = reduce(
        lambda x, y: x + y, map(
            lambda student: student[2], filter(
                lambda student: student[1] >= 17, students
            )
        )
    ) / len(students)

    return f"List of students: {students}<br>Average grade of the age 17+ students: {avg_grade}"
```

🧠 Lernprozess:

Auch hier wieder genau das gleiche wie im Vorherigen Feld. Ich konnte es im letzten schon aber auch hier hatte ich ein wenig Schwierigkeiten als mein Code ein wenig zu sehr in Richtung Unleserlichkeit kippte. Was sich aber auch durch refactoring lösen liess.

🕐 Zukünftige Schritte:

Zum Schluss werde ich auch hier versuchen in der weiteren Zukunft mehr darauf zu achten welche Strukturen ich verwenden kann und meinen Code so anpassen, dass er sich mehr an einer einzelnen Struktur haltet, um so einheitlicheren Code zu schreiben.

Endpoint für die Flask Applikation: (localhost:5000)/B4E

Lernnachweis zu Kompetenz C1X

💡 Kompetenzen:

C1G: Ich kann einige Refactoring-Techniken aufzählen, die einen Code lesbarer und verständlicher machen.

C1F: Ich kann mit Refactoring-Techniken einen Code lesbarer und verständlicher machen.

C1E:Ich kann die Auswirkungen des Refactoring auf das Verhalten des Codes einschätzen und sicherstellen, dass das Refactoring keine unerwünschten Nebeneffekte hat.

✅ Lernnachweis: Das Konzept des Refactoring und dessen Auswirkungen

In diesem Letzten Lernnachweis habe ich mich ausführlich mit dem Konzept des Refactoring und der Optimierung Bestehenden Codes auseinandergesetzt. Die Herausforderung des Refactoring liegt darin zu wissen, was man überhaupt verbessern muss/kann, da es auch an eine gewisse Menge expertise bedarf.

⚡ Funktionen:

Mir sind zu diesem Teil des Kompetenzrasters leider keine möglichen Code-Beispiele eingefallen, welche man hätte vorzeigen können oder ins Projekt einbauen können. Daher finden sie auch im Projekt nur einen Link zu dieser Doku.

🔃 Refactoring:

Es gibt verschiedene Arten des Refactoring, eine Beispiele können sein:

- Entfernen von redundantem Code (DRY → Don’t repeat yourself)
- Verwenden von Lambda ausdrücken
- Verwendung von immutablen Datenstrukturen
- Verlagerung von grossen Code-Fragmenten in kleinere Funktionen
- Verbesserung der Wiederverwendbarkeit durch Nutzung von Funktionen

Mit Hilfe von Automatischen Tests, Codereviews, TDD (test driven development) und vielen weiteren Faktoren können viele der möglichen unerwünschten Nebeneffekte auch vor dem Deployment oder dem Kompilieren gefunden und entfernt werden, was es daher auch viel sicherer macht als einfach irgendwelchen geschriebenen Code zu verwenden, von welchem man nach 2 Tagen selbst nicht mal mehr weiss was der eigentlich macht.

🧠 Lernprozess:

Ich hatte schon vor diesem Modul ein gutes Vorwissen zum Thema Refactoring und dem effizient machen von Code, jedoch war mir nie bewusst, wie viel Einfluss solche Factorings auf den Code haben können. Auch wusste ich nicht wie stark es helfen kann den Code einfacher lesbar zu machen, was ich zuvor immer eine sehr zeitaufwändige Aufgabe fand.

🕐 Zukünftige Schritte:

Um alles abzuschliessen muss ich sagen, dass ich sicher noch sehr sehr viel mit Refactoring und Verbesserung existierenden Code zu tun haben werde, sowohl in der Schule als auch im Praktikum, als auch im Arbeitsleben danach (oder so ist es mir zumindest erklärt worden). Es ist ein wichtiger Teil des Programmierens, welcher nicht vernachlässigt werden darf und vielen Personen die mit dem eigenen Code arbeiten müssen das Leben stark vereinfachen kann.

Endpoints für die Flask Applikation:

- (localhost:5000)/C1G
- (localhost:5000)/C1F
- (localhost:5000)/C1E