

Modul M323 – Funktionales Programmieren

Bank Marketing Data – Kundenanalyse & Abschlussprognose

Projekt-Dokumentation

Autoren: Peter Ngo, Alex Uscata

Klasse : INA 23A

Dozent: Dieter Kopp

Datum: 13. Dezember 2025

Inhaltsverzeichnis

1	Einleitung	2
2	Wahl der imperativen Programmiersprache	2
2.1	Begründung der Sprachwahl	2
2.2	Unterstützte funktionale Elemente in Python	2
3	Projektantrag	3
3.1	Ausgangslage	3
3.2	Zielsetzung	3
3.3	Projektumfang	3
4	Programmausgabe	3
4.1	Beispielhafter Output	3
5	Fazit	4
5.1	Nutzen funktionaler Elemente	4
5.2	Vereinfachung durch Refactoring	4
5.3	Wiederverwendung funktionaler Konzepte	4
5.4	Anwendungsfälle im beruflichen Umfeld	4

1 Einleitung

Im Rahmen des Moduls M323 – Funktionales Programmieren wird in diesem Projekt eine Datenanalyse einer Bank-Marketingkampagne umgesetzt. Ziel ist es, eine imperativ geschriebene Lösung mit einer funktional refaktorierten Version zu vergleichen und die Vor- sowie Nachteile funktionaler Sprachelemente praxisnah zu evaluieren.

Als Datengrundlage dient ein realitätsnaher Datensatz einer portugiesischen Bank, welcher Kundendaten sowie den Erfolg einer Marketingkampagne enthält. Beide Programmversionen erzeugen denselben Output, unterscheiden sich jedoch grundlegend in der Programmierweise.

2 Wahl der imperativen Programmiersprache

2.1 Begründung der Sprachwahl

Für dieses Projekt wurde die Programmiersprache **Python** gewählt. Python eignet sich besonders gut für Datenanalysen, da die Sprache:

- leicht lesbar und verständlich ist
- imperative Programmierung vollständig unterstützt
- funktionale Sprachelemente integriert anbietet
- häufig im beruflichen Umfeld eingesetzt wird

Da Python sowohl imperative als auch funktionale Konzepte vereint, ist die Sprache ideal geeignet, um beide Programmierparadigmen direkt miteinander zu vergleichen.

2.2 Unterstützte funktionale Elemente in Python

Python ist keine rein funktionale Sprache, bietet jedoch zahlreiche funktionale Sprachelemente aus der Standardbibliothek:

- `map()` – Transformation von Daten
- `filter()` – Selektion von Elementen
- `reduce()` (aus `functools`) – Aggregation
- Lambda-Funktionen
- List Comprehensions
- Unveränderliche Datentypen (z. B. Tupel)

Diese Elemente ermöglichen eine deklarative und kompakte Beschreibung von Datenverarbeitungslogik.

3 Projektantrag

3.1 Ausgangslage

Während einer Marketingkampagne hat eine Bank verschiedene Kundendaten erfasst. Der Datensatz liegt in Form einer CSV-Datei vor und ist nicht direkt für Analysen aufbereitet.

Ziel ist es, relevante Kennzahlen zu berechnen, Kundengruppen zu vergleichen und statistische Zusammenhänge zu untersuchen.

3.2 Zielsetzung

Das Projekt verfolgt folgende Ziele:

- Analyse der Erfolgsquote der Marketingkampagne
- Vergleich verschiedener Kundengruppen
- Umsetzung einer imperativen Version (V1.0)
- Refactoring in eine funktionale Version (V2.0)
- Vergleich beider Ansätze hinsichtlich Lesbarkeit und Wartbarkeit

3.3 Projektumfang

Das Projekt umfasst:

- Einlesen und Verarbeiten eines CSV-Datensatzes
- Textbasierte Konsolenausgabe
- Keine externen Bibliotheken
- Umsetzung in Python und Jupyter Notebook

4 Programmausgabe

Beide Versionen des Programms erzeugen denselben Output. Die Ausgabe erfolgt textbasiert in der Konsole.

4.1 Beispielhafter Output

```
1 =====
2 Bankkampagne      Erfolgsquote
3
4 Anzahl Kunden: 4521
5 Absch l sse (yes): 512
6 Erfolgsquote: 11.3 %
7
8 =====
```

```

9  Vergleich nach Education
10
11 primary    |      Balance: 850   | Erfolgsquote: 8.5 %
12 secondary  |      Balance: 1200  | Erfolgsquote: 10.1 %
13 tertiary   |      Balance: 1800  | Erfolgsquote: 14.3 %
14
15 ANOVA F-Wert: 5.21

```

Hinweis: Version 1.0 (imperativ) und Version 2.0 (funktional) liefern identische Resultate.

5 Fazit

5.1 Nutzen funktionaler Elemente

Der Einsatz funktionaler Sprachelemente hat in diesem Projekt mehrere Vorteile gebracht. Insbesondere Filter- und Transformationsoperationen konnten kompakter und klarer formuliert werden.

5.2 Vereinfachung durch Refactoring

Im Vergleich zur imperativen Version reduzierte sich die Code-Länge in Version 2.0 deutlich. Verschachtelte Schleifen und Hilfsvariablen konnten durch deklarative Ausdrücke ersetzt werden. Dies führte zu besserer Lesbarkeit und weniger Fehleranfälligkeit.

5.3 Wiederverwendung funktionaler Konzepte

Die funktionalen Konzepte würden in zukünftigen Projekten erneut eingesetzt werden, insbesondere bei:

- Datenfiltern
- Aggregationen
- Transformationen von Listen und Datensätzen

5.4 Anwendungsfälle im beruflichen Umfeld

Im betrieblichen Kontext eignen sich funktionale Sprachelemente besonders für:

- Datenanalyse
- Reporting
- Verarbeitung von Log- oder Kundendaten
- Automatisierte Auswertungen

Zusammenfassend lässt sich sagen, dass funktionale Programmieransätze eine sinnvolle Ergänzung zur imperativen Programmierung darstellen und in vielen Anwendungsfällen zu einer klareren und effizienteren Lösung führen.