**LINQ-Übungen**

**Arbeiten mit Objekten**

Gegeben ist eine Liste von Produkten:

class Produkt

{

public string Name { get; set; }

public double Preis { get; set; }

}

List<Produkt> produkte = new List<Produkt>

{

new Produkt { Name = "Apfel", Preis = 1.2 },

new Produkt { Name = "Banane", Preis = 0.8 },

new Produkt { Name = "Kirsche", Preis = 2.5 },

new Produkt { Name = "Mango", Preis = 3.0 },

new Produkt { Name = "Orange", Preis = 1.5 }

};

**Aufgaben:**

1. Ermittle alle Produkte, die weniger als 2 Euro kosten.
2. Sortiere die Produkte nach Preis aufsteigend.
3. Gib nur die Namen der Produkte als Liste zurück.
4. Berechne den Durchschnittspreis aller Produkte.

**Gruppierung und Aggregation**

Gegeben ist eine Liste von Personen:

class Person

{

public string Name { get; set; }

public int Alter { get; set; }

}

List<Person> personen = new List<Person>

{

new Person { Name = "Alice", Alter = 25 },

new Person { Name = "Bob", Alter = 30 },

new Person { Name = "Charlie", Alter = 25 },

new Person { Name = "David", Alter = 35 },

new Person { Name = "Eve", Alter = 30 }

};

**Aufgaben:**

1. Gruppiere die Personen nach Alter.
2. Ermittle das Durchschnittsalter aller Personen.
3. Zähle, wie viele Personen pro Altersgruppe existieren.

**Fortgeschrittene LINQ-Abfragen**

Gegeben ist eine Liste von Bestellungen:

class Bestellung

{

public int Id { get; set; }

public string Kunde { get; set; }

public double Betrag { get; set; }

}

List<Bestellung> bestellungen = new List<Bestellung>

{

new Bestellung { Id = 1, Kunde = "Müller", Betrag = 100 },

new Bestellung { Id = 2, Kunde = "Schmidt", Betrag = 200 },

new Bestellung { Id = 3, Kunde = "Müller", Betrag = 150 },

new Bestellung { Id = 4, Kunde = "Schneider", Betrag = 300 },

new Bestellung { Id = 5, Kunde = "Schmidt", Betrag = 50 }

};

**Aufgaben:**

1. Berechne den Gesamtbetrag aller Bestellungen.
2. Finde den Kunden mit dem höchsten Gesamtbestellwert.
3. Gruppiere die Bestellungen nach Kunde und berechne den Gesamtbetrag pro Kunde.

**Joins mit LINQ**

Gegeben sind zwei Listen:

class Kunde

{

public int Id { get; set; }

public string Name { get; set; }

}

List<Kunde> kunden = new List<Kunde>

{

new Kunde { Id = 1, Name = "Müller" },

new Kunde { Id = 2, Name = "Schmidt" },

new Kunde { Id = 3, Name = "Schneider" }

};

Die Liste **bestellungen** aus der vorherigen Aufgabe bleibt bestehen.

**Aufgaben:**

1. Erstelle eine Abfrage, die für jede Bestellung den Kundennamen anzeigt (inner join).
2. Erstelle eine Abfrage, die **alle** Kunden ausgibt und ihre Bestellungen anzeigt (left join).

Hier sind weitere **LINQ-Übungen**, die verschiedene Themen wie **Filtern, Projektionen, Gruppierungen, Joins und Aggregate** abdecken.

**LINQ mit Zeichenketten**

Gegeben ist eine Liste von Städten:

List<string> staedte = new List<string> { "Wien", "Graz", "Linz", "Salzburg", "Innsbruck", "Klagenfurt", "Villach", "Wels", "St. Pölten", "Dornbirn" };

**Aufgaben:**

1. Finde alle Städte, die mit „D“ beginnen.
2. Sortiere die Städte alphabetisch und gib sie aus.
3. Gib die Anzahl der Städte aus, die mehr als 6 Buchstaben haben.
4. Erstelle eine neue Liste mit allen Städten in Großbuchstaben.

**LINQ mit Datum und Zeit**

Gegeben ist eine Liste von Ereignissen:

class Ereignis

{

public string Beschreibung { get; set; }

public DateTime Datum { get; set; }

}

List<Ereignis> ereignisse = new List<Ereignis>

{

new Ereignis { Beschreibung = "Projektstart", Datum = new DateTime(2024, 1, 10) },

new Ereignis { Beschreibung = "Meilenstein 1", Datum = new DateTime(2024, 2, 15) },

new Ereignis { Beschreibung = "Meilenstein 2", Datum = new DateTime(2024, 3, 20) },

new Ereignis { Beschreibung = "Abschluss", Datum = new DateTime(2024, 4, 25) }

};

**Aufgaben:**

1. Finde alle Ereignisse, die in der Zukunft liegen.
2. Ermittle das früheste Ereignis.
3. Sortiere die Ereignisse nach Datum absteigend.
4. Zähle, wie viele Ereignisse im Monat Februar liegen.

**Komplexe Filter und Transformationen**

Gegeben ist eine Liste von Mitarbeitern:

class Mitarbeiter

{

public string Name { get; set; }

public string Abteilung { get; set; }

public double Gehalt { get; set; }

}

List<Mitarbeiter> mitarbeiter = new List<Mitarbeiter>

{

new Mitarbeiter { Name = "Anna", Abteilung = "IT", Gehalt = 4500 },

new Mitarbeiter { Name = "Bernd", Abteilung = "HR", Gehalt = 3200 },

new Mitarbeiter { Name = "Clara", Abteilung = "IT", Gehalt = 5000 },

new Mitarbeiter { Name = "David", Abteilung = "Marketing", Gehalt = 3500 },

new Mitarbeiter { Name = "Emma", Abteilung = "IT", Gehalt = 4800 }

};

**Aufgaben:**

1. Finde alle Mitarbeiter der IT-Abteilung.
2. Berechne das Durchschnittsgehalt aller Mitarbeiter.
3. Erstelle eine Liste aller Mitarbeiter mit einer Gehaltserhöhung um 10 % (aber ohne die Originalwerte zu ändern).
4. Gruppiere die Mitarbeiter nach Abteilung und berechne das höchste Gehalt pro Abteilung.

**LINQ mit Dictionaries**

Gegeben ist ein Dictionary mit Bestellnummern und Beträgen:

Dictionary<int, double> bestellungen = new Dictionary<int, double>

{

{ 1001, 150.50 },

{ 1002, 99.99 },

{ 1003, 250.00 },

{ 1004, 75.25 },

{ 1005, 300.00 }

};

**Aufgaben:**

1. Finde alle Bestellungen über 100 €.
2. Berechne den Gesamtwert aller Bestellungen.
3. Ermittle die Bestellnummer mit dem höchsten Betrag.
4. Erstelle eine Liste mit allen Bestellnummern als Zeichenketten ("Bestellung: 1001", "Bestellung: 1002", ...).

**Fortgeschrittene Joins**

Gegeben sind Kunden und Bestellungen:

class Kunde

{

public int Id { get; set; }

public string Name { get; set; }

}

List<Kunde> kunden = new List<Kunde>

{

new Kunde { Id = 1, Name = "Meier" },

new Kunde { Id = 2, Name = "Schulz" },

new Kunde { Id = 3, Name = "Becker" }

};

class Bestellung

{

public int Id { get; set; }

public int KundeId { get; set; }

public double Betrag { get; set; }

}

List<Bestellung> bestellungen = new List<Bestellung>

{

new Bestellung { Id = 101, KundeId = 1, Betrag = 250 },

new Bestellung { Id = 102, KundeId = 2, Betrag = 150 },

new Bestellung { Id = 103, KundeId = 1, Betrag = 300 },

new Bestellung { Id = 104, KundeId = 3, Betrag = 50 }

};

**Aufgaben:**

1. Erstelle eine Abfrage, die alle Bestellungen mit dem zugehörigen Kundennamen zurückgibt (inner join).
2. Zeige alle Kunden und ihre Bestellungen an (left join, auch wenn ein Kunde keine Bestellung hat).
3. Ermittle den Kunden mit der höchsten Gesamtsumme an Bestellungen.

**LINQ mit Verschachtelten Objekten**

Gegeben ist eine Liste von Kursen mit Teilnehmern:

class Teilnehmer

{

public string Name { get; set; }

}

class Kurs

{

public string Titel { get; set; }

public List<Teilnehmer> Teilnehmer { get; set; }

}

List<Kurs> kurse = new List<Kurs>

{

new Kurs { Titel = "C# Grundlagen", Teilnehmer = new List<Teilnehmer> { new Teilnehmer { Name = "Lisa" }, new Teilnehmer { Name = "Markus" } } },

new Kurs { Titel = "Datenbanken", Teilnehmer = new List<Teilnehmer> { new Teilnehmer { Name = "Markus" }, new Teilnehmer { Name = "Tom" } } },

new Kurs { Titel = "Webentwicklung", Teilnehmer = new List<Teilnehmer> { new Teilnehmer { Name = "Anna" }, new Teilnehmer { Name = "Lisa" } } }

};

**Aufgaben:**

1. Erstelle eine Liste aller Teilnehmer über alle Kurse hinweg (ohne Duplikate).
2. Ermittle alle Kurse, an denen „Lisa“ teilnimmt.
3. Finde den Kurs mit den meisten Teilnehmern.
4. Gruppiere die Teilnehmer nach der Anzahl der belegten Kurse.