# tic tac toe

**Step 1: Define the Board**

- Represent the board as a data structure, such as a list of lists or a flat list.

- The board will have 9 positions, labeled from 0-8 or represented by 'X', 'O', or empty.

**Step 2: Display the Board**

- Create a pure function to display the board.

- Pass the current board state to the function, and it will print the layout.

**Step 3: Define the Players**

- Use pure functions to allow player inputs.

- Players could be human or computer.

**Step 4: Check for Win Conditions**

- Create functions to check rows, columns, and diagonals to see if there's a winner.

- This should be done using functional tools like `map` or `any`.

**Step 5: Handle Player Input**

- Use a function to take user input and update the board state.

- Ensure inputs are valid by checking if the position is available.

**Step 6: Switch Players**

- Create a function that switches turns between players.

**Step 7: Game Loop**

- Use a recursive function or a loop to iterate until the game reaches a win or draw state.

- Each turn should display the board, take input, and check for the winner.

**Step 8: Check for Draw**

- Use a function to determine if all cells are filled, indicating a draw.

**Step 9: Main Function**

- Use a functional pipeline to combine all steps.

- The game loop should be driven by functional calls, where the state is updated and passed to the next function.

# Inventory Management System for a Grocery Store

**Project Overview**

The goal of this project is to create an Inventory Management System for a grocery store. This system will allow functionalities like adding new items, updating inventory levels, calculating total value, filtering items by category, and generating reports.

**Step-by-Step Implementation Skeleton**

**Step 1: Define Data Structures**

- Represent inventory items as a list of dictionaries. Each dictionary will contain details like `item\_name`, `category`, `quantity`, `price\_per\_unit`.

**Step 2: Sample Inventory Data**

- Create a sample list of items with relevant details (e.g., `item\_name`, `category`, `quantity`, `price\_per\_unit`).

**Step 3: Filter Items by Category**

- Write a pure function to filter items by their category.

- Use `filter` to extract all items belonging to a specific category, such as `fruits`, `vegetables`, or `dairy`.

**Step 4: Add a New Item**

- Write a function to add a new item to the inventory.

- This function should take the current inventory list and the new item, and return a new list with the added item.

**Step 5: Update Inventory Levels**

- Write a function to update inventory levels for a given item.

- Use `map` to modify the quantity of an item, returning a new list with the updated inventory.

**Step 6: Calculate Total Inventory Value**

- Write a function to calculate the total value of the inventory.

- Use `reduce` to calculate the total value by multiplying the quantity of each item by its price and summing them up.

**Step 7: Generate Inventory Reports**

- Write a function to generate reports, like:

- The total number of items.

- Items with quantity below a certain threshold.

- Categories with the highest value.

**Step 8: Functional Pipeline**

- Create a functional pipeline that allows:

- Adding new items.

- Updating existing items.

- Calculating total value.

- Generating reports.

**Step 9: Execute Actions**

- Create a function that simulates different operations:

- Add a new item to the inventory.

- Update quantities for specific items (e.g., restock or sell).

- Generate a summary report.

**Step 10: Main Function to Drive the System**

- Use a main function that utilizes the functional pipeline to:

- Add items.

- Update inventory.

- Filter items.

- Calculate statistics.

- Maintain immutability of data and return new states for each action.

# Student Grades Analysis System

**Project Overview**

The project is a "Student Grades Analysis System" that calculates grades, provides class statistics, and identifies high-performing and underperforming students using a functional programming approach.

**Step-by-Step Implementation Skeleton**

**Step 1: Define Data Structures**

- Represent each student as a dictionary containing fields like `name`, `grades` (list of integers for different subjects).

- The student data is represented as a list of these dictionaries.

**Step 2: Sample Student Data**

- Create a list of students with names and their respective grades in different subjects.

**Step 3: Calculate Average Grade for Each Student**

- Write a pure function to calculate the average grade for each student.

- Use `map` to iterate over students and calculate their average grades.

**Step 4: Filter Students by Performance**

- Write functions to filter students into categories:

- High Performers: Use `filter` to select students with an average grade above a certain threshold.

- Underperformers: Use `filter` to select students with an average grade below a specific threshold.

**Step 5: Find the Highest and Lowest Grades**

- Write a function to find the highest and lowest grades among all students.

- Use `reduce` to iterate over the student list and determine these values.

**Step 6: Calculate Class Average**

- Write a function to calculate the overall average grade for the entire class.

- Use `reduce` to sum up all student grades and divide by the total number of grades.

**Step 7: Generate a Report**

- Write a function to generate a report with the following:

- Class average.

- List of high performers.

- List of underperformers.

**Step 8: Sort Students by Grades**

- Use `sorted` and a lambda function to create a ranked list of students based on their average grades.

**Step 9: Functional Pipeline**

- Create a functional pipeline to handle operations:

- Adding new students.

- Calculating average grades.

- Filtering students by performance.

- Generating reports.

**Step 10: Execute Actions**

- Create a function that simulates different operations:

- Add a new student with their grades.

- Calculate the class average.

- Generate a summary report including high and underperforming students.

**Step 11: Main Function to Drive the System**

- Use a main function to:

- Add students to the list.

- Calculate statistics such as average, highest, and lowest grades.

- Generate reports.

- Maintain the immutability of data and ensure each action returns a new state.

# Flight Booking System for Airline

**Project Overview**

This project is a "Flight Booking System for Airline" that allows customers to search for flights, make bookings, cancel reservations, and calculate statistics like most popular routes, available seats, etc., using functional programming techniques.

**Step-by-Step Implementation Skeleton**

**Step 1: Define Data Structures**

* Represent each flight as a dictionary containing flight\_number, departure, destination, total\_seats, and available\_seats.
* Represent customers and their bookings as a list of dictionaries, each containing customer\_name, flight\_number, and status (booked or canceled).

**Step 2: Sample Flight and Booking Data**

* Create a list of flight dictionaries and customer booking dictionaries as sample data.

**Step 3: Search for Available Flights**

* Write a pure function to search for available flights based on departure and destination.
* Use filter to extract flights that match the criteria and have available seats.

**Step 4: Make a Booking**

* Write a function to make a booking for a customer.
* Use map to reduce the available\_seats count for the selected flight and add the booking to the customer list.
* Ensure immutability by returning a new state with the updated values.

**Step 5: Cancel a Booking**

* Write a function to cancel a booking for a customer.
* Use map to increase the available\_seats for the canceled flight and update the customer’s booking status.

**Step 6: Calculate Popular Routes**

* Write a function to calculate the most popular routes.
* Use reduce or map to count the bookings for each departure-destination pair.

**Step 7: Generate Booking Report**

* Write a function to generate a report including:
  + Total bookings for each flight.
  + The most popular route.
  + Number of available seats for each flight.

**Step 8: Find the Flight with Most Seats Available**

* Use the max function with a lambda to find the flight with the highest number of available seats.

**Step 9: Functional Pipeline**

* Create a functional pipeline for operations:
  + Search flights.
  + Make or cancel bookings.
  + Generate reports.
  + Calculate statistics.

**Step 10: Execute Actions**

* Write a function to simulate different operations:
  + Search for flights based on departure and destination.
  + Make a new booking.
  + Cancel an existing booking.
  + Generate summary reports.

**Step 11: Main Function to Drive the System**

* Use a main function to:
  + Add bookings.
  + Search for available flights.
  + Generate booking reports.
  + Calculate the most popular routes and flights with the highest available seats.
* Ensure all functions are pure and maintain data immutability.

# E-commerce Sales Analysis System

**Project Overview**

The "E-commerce Sales Analysis System" project processes transaction data from an e-commerce website, analyzes sales performance, calculates various metrics like total revenue, top-selling products, customer trends, and generates reports.

**Step-by-Step Implementation Skeleton**

**Step 1: Define Data Structures**

* Represent each transaction as a dictionary containing order\_id, customer\_id, product\_name, quantity, price, date.
* Represent product details as a list of dictionaries including fields like product\_name, category, and stock\_available.

**Step 2: Sample Transaction Data**

* Create a list of sample transactions, with each transaction representing a purchase made by a customer.

**Step 3: Calculate Total Revenue**

* Write a pure function to calculate the total revenue.
* Use reduce to sum up the revenue, multiplying price by quantity for each transaction.

**Step 4: Find Top-Selling Products**

* Write a function to find the top-selling products.
* Use map to create a list of products with quantities sold and then use reduce or sorted to identify the top sellers.

**Step 5: Filter Transactions by Date**

* Write a pure function to filter transactions within a specific date range.
* Use filter to create a subset of transactions that fall within the specified dates.

**Step 6: Calculate Average Order Value (AOV)**

* Write a function to calculate the Average Order Value (AOV).
* Use map to extract order values and reduce to sum them, dividing by the total number of orders.

**Step 7: Find Repeat Customers**

* Write a function to identify customers who have made multiple purchases.
* Use filter and map to identify customers with more than one transaction.

**Step 8: Generate Sales Report by Category**

* Write a function to generate a sales report by category.
* Use map to group transactions by product category and calculate the total revenue per category using reduce.

**Step 9: Sort Products by Popularity**

* Use sorted to create a ranked list of products based on the number of times they were sold.

**Step 10: Functional Pipeline**

* Create a functional pipeline that:
  + Calculates total revenue.
  + Finds top-selling products.
  + Identifies repeat customers.
  + Generates category-wise reports.

**Step 11: Execute Actions**

* Write a function to simulate different analyses:
  + Calculate total sales revenue.
  + Generate a sales report for a specific date range.
  + Find the most popular products and repeat customers.

**Step 12: Main Function to Drive the System**

* Use a main function to:
  + Execute sales analysis operations.
  + Generate reports for managers.
  + Sort and display information on top-selling products and repeat customers.
* Maintain immutability and functional purity by ensuring each function returns a new state.