Javascripts Basics

If operating arrays with functional operators feels foreign to you, it is worth watching at least the first three parts of the YouTube video series [Functional Programming in JavaScript](https://www.youtube.com/playlist?list=PL0zVEGEvSaeEd9hlmCXrk5yUyqUag-n84):

* [Higher-order functions](https://www.youtube.com/watch?v=BMUiFMZr7vk&list=PL0zVEGEvSaeEd9hlmCXrk5yUyqUag-n84)
* [Map](https://www.youtube.com/watch?v=bCqtb-Z5YGQ&list=PL0zVEGEvSaeEd9hlmCXrk5yUyqUag-n84&index=2)
* [Reduce basics](https://www.youtube.com/watch?v=Wl98eZpkp-c&t=31s)

High Order Functions (Used for Composition):

Var triple = function (x) {

return x\*3

}

Var waffle = triple

Waffle (30)

Here a function is set as n number of values which is a general function and it is working as a variable such that if you use it again anytime you can use the complete function again by just changing the parameters used in it.

1. Filter: Most basic and higher-order function is filter; Filter is a function on the array that accepts another function as its argument. Filter accepts one argument only.

var animals = [

{name: 'Hob', species: 'cat'},

{name: 'Mar', species: 'cat'},

{name: 'Bul', species: 'dog'},

{name: 'Cr7', species: 'goat'},

{name: 'Messi', species: 'goat'}

];

// Filter Function Applied

var dogs = animals.filter(function(animal){

return animal.species == ‘dog’

})

Filter will loop through each item in the array and for each item it's going to pass it [into] the callback function and When it does it will expect the callback function to return either true, or false to tell filter whether or not this item should be in the new array and After it's done. It will return the new filtered array, and that will be dogs.

**Web Forms**

Web forms are a very powerful tool for interacting with users — most commonly they are used for collecting data from users, or allowing them to control a user interface.

**GPT:**

Web forms are like digital questionnaires or fill-in-the-blank pages you find on websites. They’re a way for you to send information to a website—like signing up for a newsletter, ordering something online, or submitting a contact request.

Think of them as the online version of a paper form you’d fill out at a doctor’s office or a job application. They usually have fields (little boxes or lines) where you type in stuff like your name, email, or message. Sometimes they have buttons to click, like “submit” or “send,” which sends the info you entered to the website’s server (the behind-the-scenes computer that runs the site).

For example, when you log into a site, you’re using a web form with a username and password field. Simple, right? It’s just a tool websites use to collect info from you in an organized way.

**JASON Server**

A JSON server is a lightweight tool or application that mimics a full-fledged server by serving data in JSON format, typically for testing, prototyping, or small-scale projects. It’s not a "server" in the traditional sense (like Apache or Nginx), but rather a simple way to simulate a RESTful API without needing a complex backend. Here’s how it works:

**Core Concept**

A JSON server takes a single JSON file (or a set of files) as its "database" and exposes it through HTTP endpoints. It automatically generates routes based on the structure of that JSON data, allowing you to perform CRUD operations (Create, Read, Update, Delete) via standard HTTP methods (GET, POST, PUT, DELETE).

**Axios and promises**

* **Axios:** A tool for making HTTP requests, built on Promises, with extras like JSON handling.
* **Promises:** JavaScript’s way to manage async tasks, giving you a clean way to handle success or failure.

**What is Axios?**

Axios is a popular JavaScript library used to make HTTP requests from a browser or Node.js to a server (e.g., fetching data from an API). It’s built on top of the native fetch API but adds a bunch of conveniences.

**How It Works**

* You use Axios to send requests like GET, POST, PUT, or DELETE to an endpoint, and it returns the response data in a clean, usable format (usually JSON).

A **Promise** is a built-in JavaScript object that represents the eventual result (or failure) of an asynchronous operation—like fetching data from a server. It’s a way to handle tasks that take time without blocking your code.

**How It Works**

* A Promise has three states:
  1. **Pending**: The operation is ongoing (e.g., waiting for the server).
  2. **Fulfilled**: The operation succeeded, and you get a value (e.g., data).
  3. **Rejected**: The operation failed, and you get an error.
* You interact with it using .then() for success and .catch() for errors.