//Object Destructuring in jS

const favefilm = {

    title: "Top Gun",

    year: 1993,

    gener: "Action",

    star: "Amir Khan",

    director: "Tony stark"

}

const {title, year, gener, star, director} = favefilm

console.log(`My Fave film is ${title} starring ${star}. It is been directed by

    ${director} in year ${year}`)

//Without destructuring

const title11 = favefilm.title

const year1 = favefilm.year

const gener1 = favefilm.gener

const star1 = favefilm.star

const director1 = favefilm.director

console.log(`My Fave film is ${title1} starring ${star1}. It is been directed by

    ${director1}` in year1)

//SetTimeout and clearTimeout

console.log('start');

setTimeout(() => {

    console.log('Executed after 2 seconds');

},2000);

setTimeout(() => {

    console.log('Executed after 2 seconds again in parallel');

},2000);

setTimeout(() => {

    console.log('Executed after 4 seconds');

},4000);

const settimeoutvar = setTimeout(() => {

    console.log('Executed after 5 seconds');

},5000);

clearTimeout(settimeoutvar);

console.log('timeout after 5 seconds is clear');

console.log('End');

//Output

Start

End

Executed after 2 sec // Executed differently from the flow

Executed after 2 sec again in parallel// Executed after 2 seconds in parallel with above statement

Executed after 4 seconds

//Setinterval

function startCountdown(device) {

    let secremaining = 3

    const shutdowntimer = setInterval(

        function(){

            if (secremaining > 0)

            {

                console.log("Your device will shutdown in ${secremaining} seconds")

                secremaining--

            }

            else

            {

                console.log('Your device is shutting down')

                clearInterval(shutdowntimer)

            }

        }, 1000)

}

//StartCountdown('Macbook')

//Throw keyword error constructor

function checkUsername(userName){

    if (userName){

        console.log(userName)

    } else {

        console.log("Before executed")

        throw new Error('No username provided')

        console.log("will not be executed")

    }

}

checkUsername()

//Convert Ml to KM!

const onearray = [10, 20, 30, 40, 50]

const conversionFactorMilesToKm = 1.6

const newvariable  = onearray.map(function(variablehere, index){

    console.log(variablehere)

    return `Month ${index}: ${variablehere \* conversionFactorMilesToKm} KM`

})

//Or this can also be written as arrow function

const arraywithvariable = onearray.map((variable1) => {

    console.log(variable1);

    return variable1\*2;

})

//Map and foreach difference is foreach doesn't return new array

//We have to use push with foreach, We do have other methods such as shifts

const playlistHtml = playlistArr.map(function(track){

    return `

    <section class="card">

        <div class="card-start">

            <img src="/image/${track.albumArt}"

        </div>

            <div class="card-mid">

                <h4 class="card-title">${track.albumTitle}</h4>

                <p class="card-artist">${track.artist}</p>

            </div>

        <div class="class-end">

            <p class="card-menu">...</p>

        </div>

    </section>

    `

}).join('')

document.getElementById('container').innerHTML = playlistHtml

//filter function

const ages = [1, 4, 5, 6, 7, 8, 10, 44, 45, 56, 78, 89, 90]

const adults = ages.filter(function(age){

    if ( age > 18){

        return true

    }

    else{

        return false

    }

})

const adults = ages.filter(function(age) => {

    return age > 18 ? true : false

})

const adults = ages.filter(function(age) => {

    return age > 18

})

console.log(adults)

//Working with map filter and .includes search through array

const series = [

    {

        name: 'The Wire',

        location: 'Baltimore',

        lengthInHours : 60,

        genres: ['action', 'thriller', 'detective', 'suspense']

    },

    {

        name: 'Game of Thromes',

        location: 'Westeros and Essos',

        lengthInHours : 70.25,

        genres: ['fantasy', 'action', 'tragedy']

    },

    {

        name: 'Friends',

        location: 'New York',

        lengthInHours : 85,

        genres: ['comedy', 'romance', 'drama']

    },

    {

        name: 'The Walking Dead',

        location: 'Atlanta',

        lengthInHours : 131,

        genres: ['zombie', 'apocalypse', 'thriller', 'suspense']

    },

    {

        name: 'The Big Bang Theory',

        location: 'Pasadena',

        lengthInHours : 139.66,

        genres: ['comedy', 'nerd', 'romance']

    },

]

const newYorkSeries = series.filter(function(show){

    return show.location === 'New York'

})

console.log(newYorkSeries)

const thrillers = series.filter(function(show){

    return show.genres.includes('thriller')

})

console.log(thrillers)

/\*

Challenge:

1. Use the .filter() method to create an array

   of all of the thrillers.

   .includes('romance')

\*/

//Nested for loop example

const characters = [

    {

        title: 'Ninja',

        emoji: '🥷',

        powers: ['agility', 'stealth', 'aggression'],

    },

    {

        title: 'Sorcerer',

        emoji: '🧙',

        powers: ['magic', 'invisibility', 'necromancy'],

    },

    {

        title: 'Ogre',

        emoji: '👹',

        powers: ['power', 'stamina', 'shapeshifting'],

    },

    {

        title: 'Unicorn',

        emoji: '🦄',

        powers: [ 'flight', 'power', 'purity'],

    }

]

for (let character of characters){

    for (let power of character.powers){

        console.log(power)

    }

/\*

Challenge:

1. Nest a for of inside this for of to iterate over

   the powers array for each character. Log out each

   power.

\*/

}

//Nested ForEach  same as above example as nested for

characters.forEach(function(character){

    character.powers.forEach(function(power){

        console.log(power)

    })

/\*

Challenge:

1. Nest a forEach to log out each individual

   power in each characters powers array.

\*/

})

characters.forEach(function(character, index){

    console.log(index, character.title)

/\*

Challenge:

1. Nest a forEach to log out each individual

   power in each characters powers array.

\*/

})

// include

const variable = ['A','B','C','D']

console.log(variable.includes('Q'))

output: False

// Find below code

const addItemBtn = document.getElementById('add-item-btn')

const itemInput = document.getElementById('item-input')

const list = document.getElementById('list')

const shoppingList = []

addItemBtn.addEventListener('click', function(){

    /\*

    Challenge:

    1. Add an if else to the event listener's function.

    2. Only add an item to the shoppingList array if it

       is not already in the shoppingList array.

    3. If an item is a duplicate, clear the input field

       and log out "no duplicates".

    \*/

            if(shoppingList.includes(itemInput.value)){

                console.log('no duplicates')

            }

            else{

                shoppingList.push(itemInput.value)

                render()

            }

            itemInput.value = ''

})

function render(){

    let html = ''

    for (let item of shoppingList){

        html+= `<li class="list-item">${item}</li>`

    }

    list.innerHTML = html

}

render()

//

//Challenge

//1. Use the .join() method to remove

//   those annoying commas!

//⚠️ You will need to chain two methods

//   together to complete the challenge.

//

const playlistHtml = playlistArr.map(function(track){

    return `

    <section class="card">

        <div class="card-start">

            <img src="/images/${track.albumArt}">

        </div>

            <div class="card-mid">

                <h4 class="card-title">${track.title}</h4>

                <p class="card-artist">${track.artist}</p>

            </div>

        <div class="card-end">

            <p class="card-menu">...</p>

        </div>

    </section>

    `

}).join('')

document.getElementById('container').innerHTML = playlistHtml

// In Python

//playlist\_arr = [

//    {

//        "albumArt": "album1.jpg",

//       "title": "Song One",

//        "artist": "Artist A"

//   },

//    {

//        "albumArt": "album2.jpg",

//        "title": "Song Two",

//        "artist": "Artist B"

//    },

//    {

//        "albumArt": "album3.jpg",

//        "title": "Song Three",

//        "artist": "Artist C"

//    }

//]

//playlist\_html = "".join([

//    f"""

//    <section class="card">

//        <div class="card-start">

//            <img src="/images/{track['albumArt']}">

//        </div>

//        <div class="card-mid">

//            <h4 class="card-title">{track['title']}</h4>

//            <p class="card-artist">{track['artist']}</p>

//        </div>

//        <div class="card-end">

//            <p class="card-menu">...</p>

//        </div>

//    </section>

//    """

//    for track in playlist\_arr

//])

//

//print(playlist\_html)

/\*

Challenge

1. Use the .reduce() method to find the total of all of the students grades.

2. Do some simple maths to log out the class average.

\*/

const grades = [75, 83, 66, 43, 55, 99, 87, 16, 89, 64, 70, 80, 94, 77, 66, 73]

// in reduce accumulator + current value can find min max etc

const totalGrades = grades.reduce(function(total, currentGrade){

    return total + currentGrade

})

console.log(`The class average is ${totalGrades/grades.length}`)

// Reduce With object

export const studentsArr = [

    {

        name: 'Mike',

        grade: 75

    },

    {

        name: 'Emma',

        grade: 83

    },

    {

        name: 'Seth',

        grade: 66

    }

]

import { studentsArr } from '/studentsArr'

function calculateClassAverage(studentsArr) {

    const totalGrades = studentsArr.reduce(function(total, currentStudent){

        return total + currentStudent.grade

    }, 0)

    return totalGrades / studentsArr.length

}

console.log(calculateClassAverage(studentsArr))

// For loop with break and continue

/\*

    Below is an array of objects representing expenses and refunds.

    Positive amounts are expenses, negative amounts are refunds.

    We want to find out how much money was spent in 2023.

\*/

const expensesAndRefunds = [

    { description: 'Groceries', amount: 50, year: 2023 },

    { description: 'Electronics', amount: -30, year: 2023 },

    { description: 'Dinner', amount: 40, year: 2023 },

    { description: 'Clothing', amount: 60, year: 2023 },

    { description: 'Entertainment', amount: 25, year: 2023 },

    { description: 'Rent', amount: -500, year: 2024 },

    { description: 'Utilities', amount: 100, year: 2024 },

    { description: 'Books', amount: 20, year: 2024 },

    { description: 'Fitness', amount: 30, year: 2024 },

    { description: 'Gifts', amount: 15, year: 2024 },

]

let totalSpent = 0

const cutoffDate = 2024

for (let i = 0; i < expensesAndRefunds.length; i++) {

    const currentExpenseOrRefund = expensesAndRefunds[i]

    if (currentExpenseOrRefund.year >= cutoffDate) {

        console.log(`Reached cutoff date, exiting loop`)

        break

    }

    if (currentExpenseOrRefund.amount < 0) {

        console.log(`Skipping ${currentExpenseOrRefund.description} due to refund`)

        continue

    }

    totalSpent += currentExpenseOrRefund.amount

}

console.log(`Total amount spent on items in 2023: $${totalSpent}`)

const dailyStepsArr = [10000, 12000, 18000, 15000, 11000, 19000, 13000]

const areAllOver10K = dailyStepsArr.every(function(x) {

    return x >= 10000

})

console.log(areAllOver10K) // False

const aresome = dailyStepsArr.some(function(x) {

    return x >= 10000

})

console.log(aresome)// True

const invoiceUSDArr = [201, 354, 26, 1299, 1400, 60, 76]

const invoiceOver1k = invoiceUSDArr.find(function() {

    return invoiceOver1k > 1000

})

console.log(invoiceOver1k) //InvoiceOver1k

const invoiceIndexOver1k = invoicesUSDArr.findIndex(function(invoice) {

    return invoice > 1000

})

console.log(invoiceIndexOver1k)// 3

// .indexOf() gives us the index of a given item in the array.

console.log(invoicesUSDArr.indexOf(26))

function setPermissionLevel(permissionLevel, ...names) {

    console.log(names)

    // ["Dave", "Sally"]

    // console.log(`${name1} now has ${permissionLevel} level access.`)

    // console.log(`${name2} now has ${permissionLevel} level access.`)

    // console.log(`${name3} now has ${permissionLevel} level access.`)

}

setPermissionLevel('admin', 'Dave', 'Sally')

As it’s returning list of parameters.

function setPermissionLevel(permissionLevel, ...names) {

    names.forEach((name)=>

    console.log(`${name} now has ${permissionLevel} level access.`))

}

setPermissionLevel('admin', 'Dave', 'Sally')

function getLabelsHtml(text, sender, ...staffObjs) {

    const labelsHtml = staffObjs.map(staffObj =>

`<div class="label-card">

    <p>Dear ${staffObj.name}</p>

    <p>${text}</p>

    <p>Best wishes,</p>

    <p>${sender}</p>

</div>`

    ).join('')

    return labelsHtml

}

const text = 'Thank you for all your hard work throughout the year! 🙏🎁'

const sender = 'Tom'

document.getElementById('labels-container').innerHTML = getLabelsHtml(

    text,

    sender,

    {name: 'Sally'},

    {name: 'Mike'},

    {name: 'Rob'},

    {name: 'Harriet'}

    )

Logical && And || operators

/\*

Challenge: Refactor the Library System Code.

You are working with a piece of code that handles

information about books in a library system. The

current code is functional but can be improved to

be more concise and efficient by using short-circuiting,

nullish coalescing, and Optional Chaining.

The existing code performs the following tasks:

- Pushes an object with title, author, year published,

  and availability to an array 'collection'

- Provides default values for missing information.

\*/

const collection = []

function addBookToCollection(title, author, yearPublished, libraryData) {

    // Explicitly check for title

    title = title || 'Unknown Title'

    // Explicitly check for author

    author = author || 'Unknown Author'

    // Explicitly check if yearPublished is undefined or null

    yearPublished = yearPublished ?? 'Not Specified'

    // Explicitly check for availability at main library

    // let availability = 'Not Available' // Default value

    // if (libraryData) {

    //     if (libraryData.locations) {

    //         if (libraryData.locations.mainLibrary) {

    //             availability = 'Available'

    //         }

    //     }

    // }

    const availability = libraryData?.locations?.mainLibrary && "Available" || "Not Available"

    // Push the book object to 'collection'

    collection.push({

        title: title,

        author: author,

        yearPublished: yearPublished,

        availability: availability

    })

}

// Examples of adding a book

addBookToCollection('JavaScript: The Good Parts', 'Douglas Crockford', 2008, { locations: { mainLibrary: true}})

addBookToCollection('', '', null, null, )

addBookToCollection('House of Giants', 'Gemma Small', '', null, {locations: { mainLibrary: null}})

console.log(collection) // House of Giants is not yet published!!

const button = document.getElementById('btn')

const product = {

    name: 'Vanilla Lip Gloss',

    sku: 'w234fg',

    stock: 276,

    getProductInfo: function() {

        // console.log(this)

        console.log(`Stock level for ${this.name} (SKU: ${this.sku}): ${this.stock}`)

    }

}

/\*

Challenge 1:

  What is the 'this' value of 'product.getProductInfo' as we

  are using it now in the eventListener?

  Write your answer here: The element that was clicked!(Without bind)

Challenge 2:

  Debug the code so it works as intended.

\*/

button.addEventListener('click', product.getProductInfo.bind(product))

class Employee {

    constructor(name) {

        this.name = name

    }

    static getNewIntern(name) {

        return {

            name,

            role: 'intern',

            startDate: (new Date).toDateString()

        }

    }

/\*

Challenge:

1. Set up a static method called getNewIntern.

   getNewIntern should return an object with a 'name' property,

   a 'role' property which is hard-coded to ‘intern’, and a

   'startDate' property which should be the time of code execution.

   (🤔 How can you use JS to get the time right now?)

\*/

}

 console.log(Employee.getNewIntern('Dave'))

class Person {

    // Private fields (only accessible within the class)

    #name;

    #age;

    constructor(name, age) {

      this.#name = name;

      this.#age = age;

    }

    // Getter method for name

    getName() {

      return this.#name;

    }

    // Setter method for name

    setName(name) {

      this.#name = name;

    }

    // Getter method for age

    getAge() {

      return this.#age;

    }

    // Setter method for age

    setAge(age) {

      if (age > 0) {

        this.#age = age;

      } else {

        console.log('Age must be positive');

      }

    }

  }

  const person = new Person('John', 25);

  console.log(person.getName()); // John

  console.log(person.getAge());  // 25

  person.setName('Alice');

  person.setAge(30);

  console.log(person.getName()); // Alice

  console.log(person.getAge());  // 30

  // The following would throw an error as #name and #age are private:

  // console.log(person.#name); // Error

creation of new object in JS:-

  1. Using Object Literals

  The most straightforward way to create an object is by using object literal syntax.

  const person = {

    name: 'John Doe',

    age: 30,

    email: 'johndoe@example.com'

  };

  console.log(person);

  2. Using the new Object() Syntax

  You can also create an object using the new Object() syntax, but this is less common and not as concise as using object literals.

    const person = new Object();

    person.name = 'John Doe';

    person.age = 30;

    person.email = 'johndoe@example.com';

    console.log(person);

  3. Using a Constructor Function

  Another way to create objects is by using a constructor function, especially when you need to create multiple objects with similar properties.

  function Person(name, age, email) {

    this.name = name;

    this.age = age;

    this.email = email;

  }

  const person1 = new Person('John Doe', 30, 'johndoe@example.com');

  const person2 = new Person('Jane Smith', 25, 'janesmith@example.com');

  console.log(person1);

  console.log(person2);

  4. Using class (ES6 Syntax)

  For a more structured and modern approach, you can use ES6 classes to create objects.

  class Person {

    constructor(name, age, email) {

      this.name = name;

      this.age = age;

      this.email = email;

    }

  }

  const person1 = new Person('John Doe', 30, 'johndoe@example.com');

  const person2 = new Person('Jane Smith', 25, 'janesmith@example.com');

  console.log(person1);

  console.log(person2);

  5. Using Object.create()

  Object.create() allows you to create a new object with a specified prototype object.

  const personPrototype = {

    greet: function() {

      console.log(`Hello, my name is ${this.name}`);

    }

  };

  const person1 = Object.create(personPrototype);

  person1.name = 'John Doe';

  person1.age = 30;

  person1.email = 'johndoe@example.com';

  person1.greet();  // Output: Hello, my name is John Doe

  console.log(person1);

**Bind**:- Function in **JS** declared within objects have its own **This** so while calling it need to bind it to the object else arrow function can be declared instead of traditional function as arrow function has its own

Option 1: Bind this to the person object

  const person = {

    name: 'Alice',

    age: 30,

    greet: function() {

        console.log(`Hello, my name is ${this.name}`);

    }

};

const greetFunction = person.greet.bind(person);

greetFunction(); // Output: Hello, my name is Alice

Option 2: Use an arrow function for greet

Arrow functions do not have their own this and instead inherit this from the enclosing scope. However, this approach is limited if greet needs to refer to dynamic contexts.

const person = {

    name: 'Alice',

    age: 30,

    greet: () => {

        console.log(`Hello, my name is ${person.name}`);

    }

};

const greetFunction = person.greet;

greetFunction(); // Output: Hello, my name is Alice

Option 3: Always call the method from the object

const person = {

    name: 'Alice',

    age: 30,

    greet: function() {

        console.log(`Hello, my name is ${this.name}`);

    }

};

person.greet(); // Output: Hello, my name is Alice

**Constructor function:-**

// const gamer = {

//     name: 'Dave',

//     score: 0,

//     incrementScore: function(){

//         this.score++

//     }

// }

function Gamer(name, score) {

    this.name = name

    this.score = score

    this.incrementScore = function() {

        this.score++

    }

}

const dave = new Gamer('Dave', 0)

const sarah = new Gamer('Sarah', 2)

console.log(dave)

dave.incrementScore()

console.log(dave)

console.log(sarah)

sarah.incrementScore()

console.log(sarah)

/\*

Challenge:

1. Create a constructor function called 'Character'.

2. Give it 'name' and 'collectedItemsArr' properties.

    - 'name' should hold the character’s name.

    - 'collectedItemsArr' should hold an array of items.

       Initialise it to an empty array.

3. Create an 'addItem' method which takes in an item as

   a parameter and adds it to 'collectedItemsArr'.

4. The addItem method should log out a sentence like

   `Merlin now has: wand, map, potion`.

5. Check it’s working by creating several instances of

   Character and adding items to their arrays.

\*/

function Character(name){

    this.name = name

    this.collectedItemsArr = []

    this.addItem = function(item){

        this.collectedItemsArr.push(item)

        console.log(`${this.name} now has: ${this.collectedItemsArr.join(', ')}`)

    }

}

const wizard = new Character('Merlin')

const witch = new Character('Hermione')

wizard.addItem('wand')

wizard.addItem('map')

wizard.addItem('potion')

witch.addItem('sword')

witch.addItem('cloak of invisibility')

A **factory function** in JavaScript is a function that creates and returns an object. It's an alternative to using classes or constructors for creating objects, and it allows for more flexible and reusable object creation patterns.

**Example of a Factory Function**

Here’s a basic example:

function createPerson(name, age) {

    return {

        name: name,

        age: age,

        greet() {

            console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);

        }

    };

}

const person1 = createPerson('Alice', 30);

const person2 = createPerson('Bob', 25);

person1.greet(); // Output: Hello, my name is Alice and I am 30 years old.

person2.greet(); // Output: Hello, my name is Bob and I am 25 years old.

### Advantages of Factory Functions

1. **Encapsulation:** Factory functions can encapsulate logic for creating objects, making them reusable and modular.
2. **No this binding issues:** You don't have to worry about this like in constructor functions or classes.
3. **Dynamic properties:** Factory functions allow the creation of objects with properties or methods that can vary based on input.
4. **Private data:** You can use closures to create private variables that are not accessible outside the factory function.

function createCounter() {

    let count = 0; // Private variable

    return {

        increment() {

            count++;

            console.log(`Count is now: ${count}`);

        },

        decrement() {

            count--;

            console.log(`Count is now: ${count}`);

        },

        getCount() {

            return count;

        }

    };

}

const counter = createCounter();

counter.increment(); // Output: Count is now: 1

counter.increment(); // Output: Count is now: 2

console.log(counter.getCount()); // Output: 2 //had to write get function for this

**Example of factory function:-**

function gamer(name, score) {

    return {

        name,

        score,

        incrementScore() {

            this.score++

        }

    }

}

const alice = gamer('Alice', 10)

const dave = gamer('Dave', 3)

const ash = gamer('Ash', 9)

alice.incrementScore()

dave.incrementScore()

ash.incrementScore()

dave.incrementScore()

ash.incrementScore()

console.log(alice)

console.log(dave)

console.log(ash)

class Gamer {

    constructor(name, score) {

        this.name = name

        this.score = score

    }

    incrementScore() {

        this.score++

    }

}

const dave = new Gamer('Dave', 0)

const sarah = new Gamer('Sarah', 0)

dave.incrementScore()

console.log(dave)

console.log(sarah)

Different ways of creating objects:-

function displayPolitician(currentSituation) {

    // console.log(this)

    console.log(`${this.name} is ${this.age} years old. Current situation: ${currentSituation}.`)

}

const politician1 = {

  name: 'Carly Fowler',

  age: 40

}

const politician2 = {

  name: 'Dave Bland',

  age: 55

}

displayPolitician.apply(politician1, 'In jail for corruption')

// displayPolitician.call(politician2, 'Resigned due to incompetence')

# Inheritance with constructor functions

function Event(name, location, date) {

    this.name = name

    this.location = location

    this.date = date

    this.getDetails = function () {

        return `Event: ${this.name}, Location: ${this.location}, Date: ${this.date}`

    }

}

function Concert(name, location, date, headliner) {

    Event.call(this, name, location, date)

    this.headliner = headliner

}

Concert.prototype = Object.create(Event.prototype)

Concert.prototype.constructor = Concert

const concert = new Concert("Summer Beats", "City Stadium", "2024-07-15", "The Electrons")

console.log(concert.getDetails())

function Event(name, location, date) {

    this.name = name

    this.location = location

    this.date = date

}

Event.prototype.getDetails = function() {

    return `Event: ${this.name}, Location: ${this.location}, Date: ${this.date}`

} //To add method in above function object

function Conference(name, location, date, keynoteSpeaker) {

    Event.call(this, name, location, date)

    this.keynoteSpeaker = keynoteSpeaker

}

// Inheritance. Extending conference function to accommodate event function

Conference.prototype = Object.create(Event.prototype) // syntax to attach prototypes

Conference.prototype.constructor = Conference // syntax to correct constructor

Conference.prototype.getDetails = function() {

    const eventBasics = Event.prototype.getDetails.call(this) // syntax to extend get details functions

    return `${eventBasics} Keynote Speaker: ${this.keynoteSpeaker}`

}

/\*

Challenge:

    1. Set up a constructor for 'Conference' which

       should take in 'keynoteSpeaker' as a parameter.

    2. 'Conference' should inherit from 'Event'.

    3. 'Conference' should have its own method 'getDetails'

       which calls Event's getDetails method and returns a string

       with name, location, date, and keynote speaker.

    4. When you set up an instance of Conference and call

       getDetails you should log out:

       Event: 10 Nights of JS, Location: Scrimba HQ, Date: 2025-09-29 Keynote Speaker: Ashley Smith

       📝 The new method should be on the prototype.

\*/

const conference = new Conference("10 Nights of JS", "Scrimba HQ", "2025-09-29", "Ashley Smith")

console.log(conference.getDetails())

Inheritance with classes:-

class Event {

    constructor(name, location, date) {

        this.name = name

        this.location = location

        this.date = date

    }

    getDetails() {

        return `Event: ${this.name}, Location: ${this.location}, Date: ${this.date}`

    }

}

class Concert extends Event {

    constructor(name, location, date, headliner) {

        super(name, location, date)

        this.headliner = headliner

    }

    getDetails() {

        const eventBasics = super.getDetails()

        return `${eventBasics} Headliner: ${this.headliner}`

    }

}

const concert = new Concert("Summer Beats", "City Stadium", "2023-07-15", "The Electrons")

console.log(concert.getDetails)

**Inheritance with classes:-**

class Event {

    constructor(name, location, date) {

        this.name = name

        this.location = location

        this.date = date

    }

    getDetails() {

        return `Event: ${this.name}, Location: ${this.location}, Date: ${this.date}`

    }

}

class TennisMatch extends Event {

    constructor(name, location, date, player1, player2) {

        super(name, location, date)

        this.player1 = player1

        this.player2 = player2

    }

    getDetails() {

        const eventBasics = super.getDetails()

        return `${eventBasics} Match: ${this.player1} vs ${this.player2}`

    }

}

/\*

Challenge:

    1. Set up a class 'TennisMatch' which

       should take in 'player1' and 'player2' as parameters.

    2. 'TennisMatch' should inherit properties and methods

       from 'Event'.

    3. 'TennisMatch' should have its own method 'getDetails'

       which calls Event's getDetails method to get the

       basic details of the event. It should return this string:

       ${eventBasics} Match: ${this.player1} vs ${this.player2}

    4. Uncomment my code below to create a new instance of TennisMatch

       and call the getDetails method.

       Hint.md for help 🛟

\*/

const tennisMatch = new TennisMatch("Grand Slam Final", "Wimbledon", "2025-07-15", "J Bloggs", "B Doe")

console.log(tennisMatch.getDetails())

**Static Variables:-**

class Employee {

    static employeeCount = 0

    constructor(name) {

        this.name = name

        Employee.employeeCount++

    }

    static getEmployeeCount() {

        return Employee.employeeCount

    }

}

const employee1 = new Employee("Alice")

const employee2 = new Employee("Bob")

const employee3 = new Employee("Smith")

console.log(Employee.getEmployeeCount())

**Static Methods:-**

class Employee {

    constructor(name) {

        this.name = name

    }

    static getNewIntern(name) {

        return {

            name,

            role: 'intern',

            startDate: (new Date).toDateString()

        }

    }

/\*

Challenge:

1. Set up a static method called getNewIntern.

   getNewIntern should return an object with a 'name' property,

   a 'role' property which is hard-coded to ‘intern’, and a

   'startDate' property which should be the time of code execution.

   (🤔 How can you use JS to get the time right now?)

\*/

}

 console.log(Employee.getNewIntern('Dave'))

class Holiday {

/\*

Challenge:

    1. Make 'price' a private field.

    2. Create a getter for price which appends a $ sign

       to the front and displays it to a max of 2 decimal

       places.

    3. Create a setter for price which updates price with a

       new price.

    4. Test!

\*/

    #destination

    constructor(destination, price) {

        this.#destination = destination

        this.price = price

    }

    get destination() {

        return this.#destination

    }

    set destination(newDestination) {

        if (typeof newDestination !== 'string' || newDestination.length <= 0){

            throw new Error('Destination not valid')

        }

        this.#destination = newDestination

    }

}

const safari = new Holiday('Kenya', 1000)

console.log(safari.price)

**Getters and setter’s example:-**

class Holiday {

    #destination

    #price

    constructor(destination, price) {

        this.#destination = destination;

        this.#price = price;

    }

    // Getter for destination (renamed to 'getDestination')

    get getDestination() {

        return this.#destination;

    }

    // Setter for destination (renamed to 'setDestination')

    set setDestination(newDestination) {

        if (typeof newDestination !== 'string' || newDestination.length <= 0) {

            throw new Error('Destination not valid');

        }

        this.#destination = newDestination;

    }

    // Getter for price (renamed to 'getPrice')

    get getPrice() {

        return `$${this.#price.toFixed(2)}`;

    }

    // Setter for price (renamed to 'setPrice')

    set setPrice(newPrice) {

        if (typeof newPrice !== 'number' || newPrice < 0) {

            throw new Error('Price must be a valid positive number');

        }

        this.#price = newPrice;

    }

}

// Test the implementation

const safari = new Holiday('Kenya', 1000.65478393);

// Set a new price using the updated setter method

safari.setPrice = 2345.456282764;

// Get the updated price using the new getter method

console.log(safari.getPrice); // Output: $2345.46

// Change the destination

safari.setDestination = 'South Africa';

console.log(safari.getDestination); // Output: South Africa

**Topics covered static variables , static methods and private variables and private methods. Make sure to have a look at the getstatus() function below as the names are pointing to the get/set functions.**

class Character {

    static count = 0

    #health = 100

    constructor(name) {

        this.name = name

        Character.incrementCount()

    }

    static incrementCount() {

        Character.count++

    }

    static getCount() {

        return Character.count

    }

    get health() {

        return this.#health

    }

    set health(value) {

        this.#health = Math.max(0, value)

    }

    takeDamage(amount) {

        this.health -= amount

    }

    get isAlive() {

        return this.#health > 0

    }

    getStatus() {

        return `${this.name} has ${this.health} health and is ${this.isAlive ? 'alive' : 'dead'}.`

    }

}

class Hero extends Character {

    constructor(name) {

        super(name)

        this.items = []

    }

    pickUpItem(item) {

        this.items.push(item)

    }

    getItems() {

        return `${this.name} has the following items: ${this.items.join(", ")}`

    }

}

class Villain extends Character {

    constructor(name, threat) {

        super(name)

        this.threat = threat

    }

    getThreat() {

        return this.threat

    }

}

// Example Usage

const merlin = new Hero("Merlin")

const medea = new Hero("Medea")

const troll = new Villain("Troll", 'I will destroy your soul!')

console.log(troll.getThreat()) // I will destroy your soul!

merlin.pickUpItem("Sword")

merlin.takeDamage(15)

medea.takeDamage(5)

medea.pickUpItem("Shield")

console.log(merlin.getItems()) // Merlin has the following items: Sword

console.log(medea.getItems()) // Medea has the following items: Shield

troll.takeDamage(101)

console.log(troll.getStatus()) //Troll has 0 health and is dead.

console.log(medea.getStatus()) // Medea has 95 health and is alive.`

console.log(merlin.getStatus()) // Merlin has 85 health and is alive.

console.log(`Total characters created: ${Character.getCount()} `) // Total characters created: 3

**Map Object Challenge:-**

const athlete1 = { name: 'Alice', averageTime10KmMins: 58.3 }

const athlete2 = { name: 'Dave', averageTime10KmMins: 53.2 }

const athlete3 = { name: 'Micky', averageTime10KmMins: 64.5 }

const athlete4 = { name: 'Judy', averageTime10KmMins: 66.0 }

/\* Challenge \*/

/\* 1. Create a map object "athletes" to store the athletes. \*/

const athletes = new Map() // Dictionary in python

function addAthlete(athlete, time) {

    /\* 2. This function should add athletes to the "athletes" map. \*/

    athletes.set(athlete, time)

}

function getSummary() {

    /\* This function should make the following appear in the console \*/

    athletes.forEach((value, key) => console.log(`${key.name}'s average time is ${key.averageTime10KmMins} but today ${key.name} achieved ${value}`))

    //Alice's average time is 58.3 but today Alice achieved 57.3

    //Dave's average time is 53.2 but today Dave achieved 61.1

    //Micky's average time is 64.5 but today Micky achieved 59.9

    //Judy's average time is 66 but today Judy achieved 61.6

}

addAthlete(athlete1, 57.3)

addAthlete(athlete2, 61.1)

addAthlete(athlete3, 59.9)

addAthlete(athlete4, 61.6)

getSummary()

|  |  |  |
| --- | --- | --- |
| **Feature** | **Map** | **Set** |
| **Purpose** | Key-value pair collection | Unique value collection |
| **Key Types** | Any type | Not applicable |
| **Value Types** | Any type | Any type |
| **Duplicates** | Keys must be unique | Values must be unique |
| **Iteration Order** | Insertion order | Insertion order |
| **Methods** | set, get, has, delete, clear | add, has, delete, clear |

function outerFunction(param) {

    const outerVariable = 'I am from the outer function'

    function innerFunction() {

        console.log(param)

        console.log(outerVariable)

    }

    return innerFunction

}

const closure = outerFunction('I am an argument')

closure()

closure()

function scorePoint(playerName) {

    let score = 0

    return function() {

        score++

        console.log(`${playerName} has ${score} points`)

    }

}

const player1 = scorePoint('Vicky')

const player2 = scorePoint('Leo')

player1()

player1()

player1()

player2()

**Closures:-**

function scorePoint(playerName) {

    let score = 0

    return {

        displayScore: function() {

            console.log(`${playerName} has ${score} points`)

        },

        incrementScore: function() {

            score++

        },

        decrementScore: function() {

            score--

        }

/\*

Challenge:

   1. Add two more functions to this object. One to increment

      the score and one to decrement the score. Call those functions

      a couple of times for each player and then call displayScore

      to check it’s working.

\*/

    }

}

const player1 = scorePoint('Vicky')

const player2 = scorePoint('Leo')

player2.incrementScore()

player2.incrementScore()

player2.decrementScore()

player1.incrementScore()

player2.decrementScore()

player1.incrementScore()

player1.displayScore()

player2.displayScore()

**Set Objects:-**

const wishListArr = ['shoes', 'after shave', 'tesla', 'after shave', 'shoes']

const wishListSet = new Set(wishListArr)

wishListSet.add('diary')

// wishListSet.delete('shoes')

console.log(wishListSet.has('shoes'))

wishListSet.forEach((listItem) => console.log(listItem))

/\* Challenge:

    1. Refactor this code to use a Set instead

       of an array so no tags can be duplicated!

\*/

const postTags = new Set()

function addTag(newTag) {

    postTags.add(newTag)

}

addTag('history')

addTag('romans')

addTag('sociology')

addTag('history')

addTag('history')

postTags.forEach((tag) => console.log(tag)) //No duplicates

**Closures:-**

/\*

Challenge:

    1. Write a function that simulates a simple bank account.

       The function should store the balance and enable users

       to deposit money, withdraw money, and check the balance.

Requirements:

- The account balance should start at zero.

- The function should return an object containing

  3 functions. One for depositing money, one for

  withdrawing money, and a getBalance function which

  logs the balance and the account holder's name.

- Deposits should add to the balance, and withdrawals should subtract

  from it.

\*/

function createBankAccount(name) {

  let balance = 0

  return {

    deposit: function(amount) {

      balance += amount

    },

    withdraw: function(amount) {

      balance -= amount

    },

    getBalance: function(){

      console.log(`${name} has £${balance} in their account`)

    }

  }

}

const daveAccount = createBankAccount('dave')

const wendyAccount = createBankAccount('wendy')

daveAccount.deposit(100)

daveAccount.withdraw(50)

wendyAccount.deposit(200)

wendyAccount.withdraw(60)

daveAccount.getBalance()

wendyAccount.getBalance()

**Recursion:-**

function countdown(count) {

    console.log(count)

    if (count <= 0) {

        return count

    }

    countdown(count - 1)

}

countdown(5)

function countUp(start, end) {

    console.log(start)

    if (start === end) {

        return start

    } else {

        countUp(start + 1, end)

    }

}

countUp(2, 6)

let runningTotal = 5

function sumToN(n) {

    if (n <= 0) {

        return 0

    } else {

        console.log(runningTotal)

        runningTotal += n-1

        return n + sumToN(n -1)

    }

}

console.log(sumToN(5))

// ›5

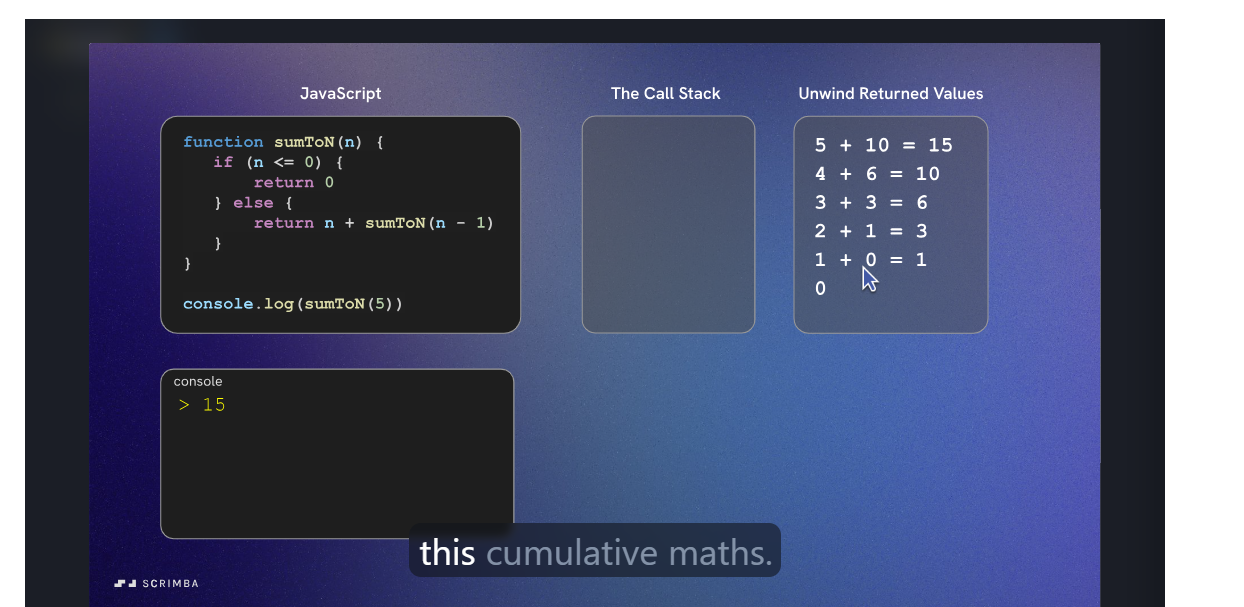
// ›9

// ›12

// ›14

// ›15

// ›15



let str = 'SCRIMBA'

function reverseStr(str) {

    if (str.length === 0) {

        return str

    } else {

        return reverseStr(str.slice(1)) + str.slice(0, 1)

    }

Here even the placement of recursion function matters as it’s been placed at the beginning of the return statement it’s resulting into reverse concat.

Output explanation:-

reverseStr(‘CRIMBA’) + S

reverseStr(‘RIMBA’)+ C + S

reverseStr(‘IMBA’)+ R + C + S

reverseStr(‘MBA’)+ I + R + C + S

reverseStr(‘BA’)+ M + I + R + C + S

reverseStr(‘A’)+ B + M + I + R + C + S

A + B + M + I + R + C + S

Final Output:- ABMIRCS

/\*

Challenge:

    1. Write logic for a recursive function

       that reverses a string.

    🛟 hint.md for help!

\*/

}

console.log(reverseStr(str))

// str.slice(start, end)

// What is the base case? The strings length being zero.

// SCRIMBA

// CRIMBA

// RIMBA

// IMBA

// MBA

// BA

// A

// ""

// A

// AB

// ABM

// ABMI

// ABMIR

// ABMIRC

// ABMIRCS

Currying:-

Original Function:-

function calculateVolume(length, width, height) {

    return length \* width \* height

}

const volume = calculateVolume(2, 3, 4)

console.log(volume)

Curry function(Arrow function syntax):-

const calculateVolume = length => width => height => length \* width \* height

// const withLength = calculateVolume(2)

// const withLengthAndWidth = withLength(3)

// const volume = withLengthAndWidth(4)

const volume = calculateVolume(2)(3)(4)

console.log(volume)

Partial application of curry functions:-

    const calculateVolume = length => width => height => length \* width \* height

    const calculateBaseAreaVolume = calculateVolume(2)(3)

    const totalVolume1 = calculateBaseAreaVolume(4)

    const totalVolume2 = calculateBaseAreaVolume(6)

    const totalVolume3 = calculateBaseAreaVolume(10)

    console.log(totalVolume1)

    console.log(totalVolume2)

    console.log(totalVolume3)

Original Functions:-

/\*

Challenge:

    1. Curry this function!

    Set up three partially applied functions called 'infoLogger',

    'warnLogger', and 'errorLogger'.

    The partially applied functions should have their level ('info',

    'warn', 'error').

    You should be able to call these functions and pass in a message.

    E.g. console.log(warnLogger("Low disk space")) would log: "[WARN] Low disk space".

    🛟 hint.md for help.

\*/

const logMessage = (level, message) => `[${level.toUpperCase()}] ${message}`

// Log messages with "info" level

console.log(logMessage("info", "Application started")) // Output: "[INFO] Application started"

console.log(logMessage("info", "User logged in")) // Output: "[INFO] User logged in"

// Log messages with "warn" level

console.log(logMessage("warn", "Low disk space")) // Output: "[WARN] Low disk space"

console.log(logMessage("warn", "High memory usage")) // Output: "[WARN] High memory usage"

// Log messages with "error" level

console.log(logMessage("error", "Unhandled exception")) // Output: "[ERROR] Unhandled exception"

console.log(logMessage("error", "Failed to save file")) // Output: "[ERROR] Failed to save file"

Currying:-

/\*

Challenge:

    1. Curry this function!

    Set up three partially applied functions called 'infoLogger',

    'warnLogger', and 'errorLogger'.

    The partially applied functions should have their level ('info',

    'warn', 'error').

    You should be able to call these functions and pass in a message.

    E.g. console.log(warnLogger("Low disk space")) would log: "[WARN] Low disk space".

    🛟 hint.md for help.

\*/

const createLogger = level => message => `[${level.toUpperCase()}] ${message}`

const infoLogger = createLogger('info')

const warnLogger = createLogger('warn')

const errorLogger = createLogger('error')

// Log messages with "info" level

console.log(infoLogger("Application started")) // Output: "[INFO] Application started"

console.log(infoLogger("User logged in")) // Output: "[INFO] User logged in"

// Log messages with "warn" level

console.log(warnLogger("Low disk space")) // Output: "[WARN] Low disk space"

console.log(warnLogger("High memory usage")) // Output: "[WARN] High memory usage"

// Log messages with "error" level

console.log(errorLogger("Unhandled exception")) // Output: "[ERROR] Unhandled exception"

console.log(errorLogger("Failed to save file")) // Output: "[ERROR] Failed to save file"

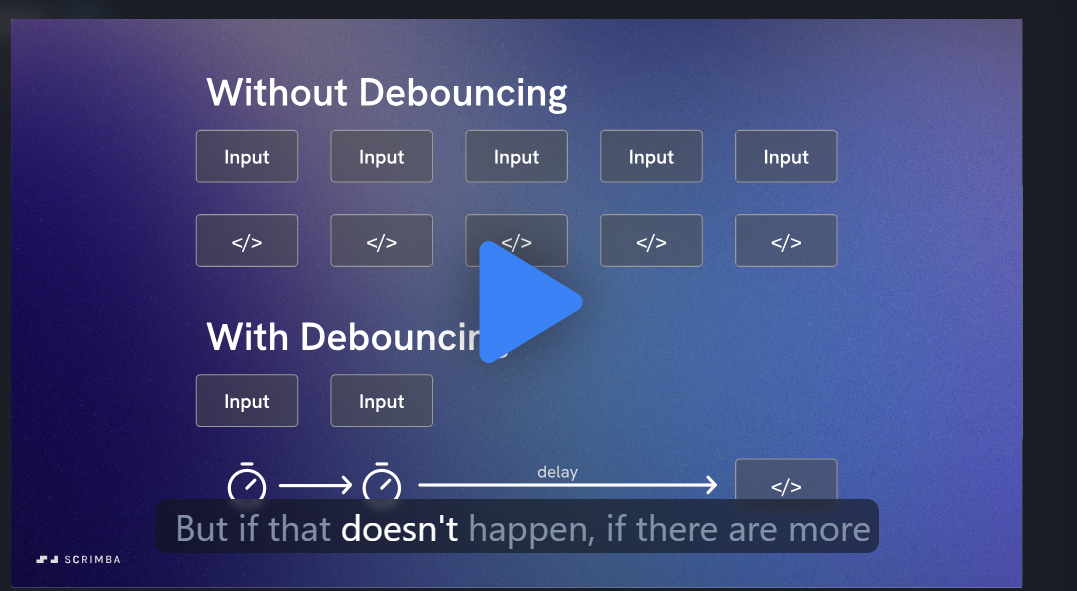
## **Throttling**

### ****What is Throttling?****

Throttling ensures a function is executed at most once in a specified time interval, no matter how often the event occurs.



**Debouncing:** Debouncing delays the execution of a function until after a specified period of inactivity. If the event is triggered again within that period, the timer resets.



# Throttling

function handleResize(e){

    console.log('resize happened on event: ' + e)

}

function throttle(func, delay) {

    let throttleTimeout = null

    return function() {

        if(!throttleTimeout) {

            func()

            throttleTimeout = setTimeout(() => {

                throttleTimeout = null

            }, delay)

        }

    }

}

const throttledHandleResize = throttle(handleResize, 1000)

window.addEventListener('resize', throttledHandleResize)

Examples:-

Original Function without debounce implemented:-

function handleInput(e) {

    console.log('Input detected from element with id ' + e.target.id)

}

// document.getElementById('name-input').addEventListener('input', debounce(handleInput, 500))

document.getElementById('name-input').addEventListener('input', handleInput)

Debounce Implemented :-

function debounce(func, delay) {

    let debounceTimer

    return function() {

        clearTimeout(debounceTimer)

        debounceTimer = setTimeout(()=> {

            func.apply(this, arguments)

        }, delay)

    }

/\*

Challenge:

    1. Write logic inside this function to make

       handleInput fire only when there has been no

       activity in the input field for ‘delay’ seconds.

       handleInput should NOT run when the first event

       is detected.

       🛟 hint.md for help!

\*/

}

function handleInput(e) {

    console.log('Input detected from element with id ' + e.target.id)

}

document.getElementById('name-input').addEventListener('input', debounce(handleInput, 500))

function debounce(func, delay) {

    let debounceTimer

    return (...args)=> {

        clearTimeout(debounceTimer)

        debounceTimer = setTimeout(()=> {

            func(...args)

        }, delay)

    }

/\*

Challenge:

    1. Convert this inner function to an arrow function and

       use rest params to handle passing in arguments.

\*/

}

function handleInput(e) {

    console.log('Input detected from element with id ' + e.target.id)

}

document.getElementById('name-input').addEventListener('input', debounce(handleInput, 500))

**Generators:-**

const slidesArr = [

    "1. Intro Slide",

    "2. The current situation",

    "3. Setbacks",

    "4. Plans",

    "5. A Positive Future"

]

function\* generator(arr) {

    for (const item of arr) {

        yield item

    }

}

const slideGenerator = generator(slidesArr)

document.getElementById('nextSlideBtn').addEventListener('click', () => {

    const result = slideGenerator.next()

    if (!result.done) {

        console.log(result.value)

    } else {

        console.log('That is the end of the presentation!')

    }

})

// Replce in JS

const paragraph = "This is the para that we have created. It was created in 1993. before Js wesites were so boring."

console.log(paragraph.replace('js', 'JS'))// First Ocurrence

console.log(paragraph.replaceAll('js', 'JS'))// ReplaceAll

//contactsArr

export const contactsArr = [

    { name: 'John Smith', email: 'john.smith@gmail.com', phone: '415-792-6341' },

    { name: 'Kofi Mensah', email: 'kofi.mensah@yahoo.com', phone: '+233-24-597-1823' },

    { name: 'Jane West', email: 'jane.west@hotmail.com', phone: '310-453-8765' },

    { name: 'Alice Johnson', email: 'alice.johnson@outlook.com', phone: '202-555-8472' },

    { name: 'Amina Hassan', email: 'amina.hassan@gmail.com', phone: '+254-712-439582' },

    { name: 'Jane Smith', email: 'jane.smith@yahoo.com', phone: '212-555-2394' },

    { name: 'Chun Hei Park', email: 'chunhei.park@naver.com', phone: '+82-10-8243-6510' },

    { name: 'Rina Nakamura', email: 'rina.nakamura@jp.com', phone: '+81-90-2746-3951' }

]

//index.js

import { contactsArr } from '/contactsData.js'

/\*

Challenge:

1. Wire up this search pattern app so that inputting

   a full or partial name brings up the matching

   contact or contacts.

\*/

const patternSearchInput = document.getElementById('pattern-search-input')

const patternSearchSubmit = document.getElementById('pattern-search-submit')

patternSearchSubmit.addEventListener('click', function() {

    findMatchingContacts(contactsArr, patternSearchInput.value)

})

function findMatchingContacts(contactsArr, pattern) {

    const regex = new RegExp(pattern, 'i')

    contactsArr.filter(function(contact) {

        return regex.test(contact.name)

    })

    .forEach(function(contact) {

        renderContact(contact)

    })

}

function renderContact(contactObj) {

    const { name, email, phone } = contactObj

    const contactDisplay = document.getElementById('contact-display')

    const contactCard = document.createElement('aside')

    contactCard.classList.add('contact-card')

    const nameElem = document.createElement('p')

    const emailElem = document.createElement('p')

    const phoneElem = document.createElement('p')

    nameElem.innerText = name

    emailElem.innerText = email

    phoneElem.innerText = phone

    contactCard.appendChild(nameElem)

    contactCard.appendChild(emailElem)

    contactCard.appendChild(phoneElem)

    contactDisplay.appendChild(contactCard)

/\*

    The CSS for contact-card has been done for you.

    The name, email and phone details can be placed in

    'p' elements and placed inside contact-card.

\*/

}

//index.html

<!doctype html>

<html>

<head>

    <title>Contacts Search</title>

    <link rel="stylesheet" href="index.css">

</head>

<body>

    <main>

        <label for="pattern-search-input">

            Enter contact name:

        </label>

        <div class="input-area">

            <input type="text" id="pattern-search-input">

            <button id="pattern-search-submit">Search</button>

        </div>

        <hr>

        <section id="contact-display" class="contact-display"></section>

    </main>

    <script src="index.js" type="module"></script>

</body>

</html>

html, body {

    margin: 0;

    padding: 0;

    color: white;

    font-family: Verdana, Geneva, Tahoma, sans-serif;

}

main {

    width: 250px;

    border: 1px solid;

    margin: 2em auto;

    padding: 1.5em;

    min-height: 400px;

    background-color: #282828;

    border-radius: 6px;

}

label {

    display: block;

    margin-bottom: .5em;

    font-size: 110%;

}

.input-area {

    display: flex;

    justify-content: center;

    margin-bottom: .8em;

}

input[type="text"]{

    width: 100%;

    border-bottom-left-radius: 5px;

    border-top-left-radius: 5px;

    border: none;

    padding: .4em;

}

button {

    border: none;

    border-bottom-right-radius: 5px;

    border-top-right-radius: 5px;

    background-color: deeppink;

    color: white;

    padding: .5em;

    letter-spacing: .05em;

}

hr {

    border: 1px solid deeppink;

}

.contact-display {

    display: flex;

    flex-direction: column;

    gap: .5em;

}

.contact-card {

    background-color: whitesmoke;

    color: #222;

    padding: 1em;

    border-radius: 6px;

}

p {

    margin: 0 0 .8em 0;

}

p:last-child {

    margin-bottom: 0px;

}

// Property For DalerArr

const propertyForSaleArr = [

    {

        propertyLocation: 'Kensington, London',

        priceGBP: 890000,

        roomsM2: [14, 18, 14, 10, 6],

        comment: 'Highly desirable location in stunning scenery!',

        image: 'cottage.jpg'

    },

    {

        propertyLocation: 'Wirral, Liverpool',

        priceGBP: 650000,

        roomsM2: [18, 16, 15, 14, 17, 19, 9, 8],

        comment: 'Astonishing view with a modern finish!',

        image: 'desres.jpg'

    },

    {

        propertyLocation: 'Beach, Brighton',

        priceGBP: 420000,

        roomsM2: [5],

        comment: 'Beautiful interior and a spacious room.',

        image: 'hut.jpg'

    },

    {

        propertyLocation: 'Highlands, Scotland',

        priceGBP: 550000,

        roomsM2: [6, 12, 11, 5],

        comment: 'Lots of potential, snug, a must see!',

        image: 'shed.jpg'

    }

]

//index.html

<!doctype html>

<html>

    <head>

        <title>Houses for Sale</title>

        <link rel="stylesheet" href="index.css">

        <link rel="preconnect" href="https://fonts.googleapis.com">

        <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

        <link href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;700&display=swap" rel="stylesheet">

    </head>

    <body>

        <section class="container" id="container"></section>

        <script src="index.js" type="></script>

    </body>

</html>

##Super Challenge: Real Estate- solution provided as below

import propertyForSaleArr from '/properties/propertyForSaleArr'

import placeholderPropertyObj from '/properties/placeholderPropertyObj'

// Note the below list

function getPropertyHtml(propertyArr = [placeholderPropertyObj]) {

    return propertyArr.map(property => {

        const { propertyLocation, priceGBP, roomsM2, comment, image } = property

        const totalRoomSizeM2 = roomsM2.reduce((total, current) => total + current)

        return `

    <section class="card">

        <img src="/images/${image}">

        <div class="card-right">

            <h2>${propertyLocation}</h2>

            <h3>£${priceGBP}</h3>

            <p>${comment}</p>

            <h3>${totalRoomSizeM2} m&sup2;</h3>

        </div>

    </section>`

    })

}

/\*\*\*\*\* Modify 👇 by adding an argument to the function call ONLY. \*\*\*\*\*/

document.getElementById('container').innerHTML = getPropertyHtml()

/\*

SUPER CHALLENGE 💪

Render out a card for each of the properties in the propertyForSaleArr array (in the 'properties' folder). Each card should have an image, a property location, a price, a comment and the TOTAL property size in square metres (each object has an array with the size in square metres of the individual rooms).

If no array of properties is passed to getPropertyHtml, the placeholder property stored in placeholderPropertyObj (in the 'properties' folder) should be rendered instead.

This is the JS I want you to use to complete this challenge 👇

- import/export

- .map()

- .join()

- Object destructuring

- .reduce()

- Default parameters

The HTML and CSS have been done for you.

This is the HTML template 👇. Replace everything in UPPERCASE with property data.

<section class="card">

    <img src="/images/IMAGE">

    <div class="card-right">

        <h2>PROPERTY LOCATION</h2>

        <h3>PRICE GBP</h3>

        <p>COMMENT</p>

        <h3>TOTAL SIZE IN SQUARE METRES m&sup2;</h3>

    </div>

</section>

\*/

//Simple Todo script without react in jS

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Todo App Without React</title>

  <style>

    button {

      margin: 5px;

    }

    hr {

      margin: 20px 0;

    }

  </style>

</head>

<body>

  <div id="root"></div>

  <script>

    // Main App function

    function App() {

      // State management

      let count = 0;

      const todos = [];

      // DOM elements

      const root = document.getElementById('root');

      // Render function to update the UI

      function render() {

        root.innerHTML = '';

        // Todos Section

        const todosSection = document.createElement('div');

        const todosHeader = document.createElement('h3');

        todosHeader.textContent = 'Todos';

        const todosList = document.createElement('ul');

        todos.forEach((todo) => {

          const listItem = document.createElement('li');

          listItem.textContent = todo;

          todosList.appendChild(listItem);

        });

        const addTodoButton = document.createElement('button');

        addTodoButton.textContent = 'Add Todo';

        addTodoButton.onclick = addTodo;

        todosSection.appendChild(todosHeader);

        todosSection.appendChild(todosList);

        todosSection.appendChild(addTodoButton);

        // Count Section

        const countSection = document.createElement('div');

        const countText = document.createElement('span');

        countText.textContent = `Count: ${count}`;

        const incrementButton = document.createElement('button');

        incrementButton.textContent = '+';

        incrementButton.onclick = increment;

        countSection.appendChild(countText);

        countSection.appendChild(incrementButton);

        // Append everything to the root

        root.appendChild(todosSection);

        root.appendChild(document.createElement('hr'));

        root.appendChild(countSection);

      }

      // Handlers

      function increment() {

        count += 1;

        render();

      }

      function addTodo() {

        todos.push('New Todo');

        render();

      }

      // Initial render

      render();

    }

    // Initialize the app

    App();

  </script>

</body>

</html>

//Same script in React

import { useState } from "react";

import ReactDOM from "react-dom/client";

import Todos from "./Todos";

const App = () => {

  const [count, setCount] = useState(0);

  const [todos, setTodos] = useState([]);

  const increment = () => {

    setCount((c) => c + 1);

  };

  const addTodo = () => {

    setTodos((t) => [...t, "New Todo"]);

  };

  return (

    <>

      <Todos todos={todos} addTodo={addTodo} />

      <hr />

      <div>

        Count: {count}

        <button onClick={increment}>+</button>

      </div>

    </>

  );

};

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(<App />);    write without React

try {

    const response = await fetch('https://apis.scrimba.com/jsonplaceholder/posts',

        {

            method: 'POST',

            body: JSON.stringify({

                title: 'Holiday Nightmares',

                body: 'When I was kidnapped in Scotland…',

                userId: 100

            }),

            headers: {

                'Content-Type': 'application/json'

            }

        })

    if (!response.ok) {

        throw new Error('There was a problem with the API')

    }

    const data = await response.json()

    console.log(data)

} catch (err) {

    console.log(err)

}

/\*

Challenge:

1. Add a headers object, setting the "Content-Type" to "application/json".

\*/</head>

-----------------

/\*

Challenge:

    1. Make a fetch request to the "Bored" API:

         Base URL: https://apis.scrimba.com/bored/api

         Endpoint: /activity

    2. Log an object containing an activity suggestion

       to the console.

    ⚠️ Make sure you use the async/await method!

    🛟 hint.md for help!

\*/

async function getSuggestion() {

    const response =  fetch('https://apis.scrimba.com/bored/api/activity')

    // const data = await response.json()

    console.log(response)

}

getSuggestion()

---------------------------------

// fetch('https://api.scrimba.com/dog.ceo/api/breeds/image/random')

//     .then(response => response.json())

//     .then(data => console.log(data))

//     .catch(err => {

//         console.log(err)

//         // update the DOM to warn the user

//         // access an alternative API

//     })

//     .finally(()=> console.log('The operation completed.'))

/\*

Challenge:

1. Convert the above code to use async/await, handle errors with “try/catch” blocks, and add a “finally” block.

\*/

try {

    const response = await fetch('https://apis.scrimba.com/dog.ceo/api/breeds/images/random')

    const data = await response.json()

    console.log(data)

} catch (err) {

    console.log(err)

    // update the DOM to warn the user

    // access an alternative API

    throw new Error('This is a network error!')

} finally {

    console.log('The operation completed.')

}

-------------------

try {

    const response = await fetch('https://apis.scrimba.com/dog.ceo/api/breeds/images/random')

    if (!response.ok){

        throw new Error('There was a problem with the API')

    }

    const data = await response.json()

    console.log(data)

} catch (err) {

    console.log(err)

    // update the DOM to warn the user

    // access an alternative API

} finally {

    console.log('The operation completed.')

}

--------------------------------

const promise = new Promise((resolve, reject)=> {

    const success = Math.random() > 0.5

    if (success) {

        resolve('Operation successful')

    } else {

        reject('Operation failed')

    }

})

// promise.then(response => console.log(response))

try {

    const response = await promise

    console.log(response)

} catch(err) {

    console.log(err)

}

------

function preLoadImg(url) {

    return new Promise((resolve, reject)=> {

      const img = new Image()

      img.src = url

      img.alt = "a beautiful scene"

      img.addEventListener('load', ()=> resolve(img))

      img.addEventListener('error', ()=> reject('img has NOT loaded'))

    })

  /\*

  Challenge:

  1. Return a new promise. The promise should:

      - create a new image and assign the incoming url

        to its src attribute. (Use the Image constructor

        for this!)

      - listen out for a load event. If a load event is

        detected, the promise should resolve, providing the

        image element.

      - listen out for an “error” event. If an error

        event is detected, the promise should reject giving

        the message “img has NOT loaded”.

  \*/

  }

  try {

    const results = await preLoadImg('https://scrimba.ams3.cdn.digitaloceanspaces.com/assets/courses/gadvancedjs/scenic1.jpg')

    console.log(results)

    document.getElementById('img-container').appendChild(results)

  } catch (error) {

    console.error(error)

  }

  ----------------

  //Callback Hell

  // upload a file

function uploadFile(callback){

    console.log('Step 1: Uploading file...')

    setTimeout(()=> {

        callback() // call next function

    }, 1000)

}

// process a file

function processFile(callback){

    console.log('Step 2: Processing file...')

    setTimeout(()=> {

        callback() // call next function

    }, 1000)

}

// notify a user

function notifyUser(callback){

    console.log('Step 3: Notifying user...')

    setTimeout(()=> {

        callback() // call next function

    }, 1000)

}

uploadFile(()=> {

    processFile( ()=> {

        notifyUser( ()=> {

            console.log('All steps completed!')

        })

    })

})

// expected output:

// Step 1: Uploading file...

// Step 2: Processing file...

// Step 3: Notifying user...

// All steps completed!

/\*

Challenge:

1. Try to invoke these functions so they run in sequence.

   The callback function passed to the final function (notifyUser)

   can just be an anonymous function that logs 'All steps completed!'

   You will come up against a gotcha here!

   hint.md is here to help!

\*/</html>

try {

    await uploadFile()

    await processFile()

    await notifyUser()

    console.log('All steps completed!')

} catch(err) {

    console.log(err)

}

----------------

function getImagePromise(url) {

    return new Promise((resolve, reject) => {

        setTimeout(() => {

            const img = new Image()

            img.src = url

            img.alt = 'scenic image'

            img.addEventListener('load', () => resolve(img))

            img.addEventListener('error', () => reject(new Error(`Failed to load image: ${url}`)))

        }, 500)

    })

}

const images = [

    'https://scrimba.com/links/advancedjs-resources-images-scenic1',

    'https://scrimba.com/links/advancedjs-resources-images-scenic2',

    'https://scrimba.com/links/advancedjs-resources-images-scenic3'

]

async function preloadImages(imageUrlsArr) {

    const imgContainer = document.getElementById('img-container')

    const uploadContainer = document.getElementById('upload-container')

    const promises = imageUrlsArr.map((url) => getImagePromise(url))

    console.log(promises)

    try{

        const results = await Promise.all(promises)

        console.log('All images loaded successfully!')

        uploadContainer.style.display = 'none'

        results.forEach((img) => imgContainer.appendChild(img))

    } catch(err) {

        console.log(err)

    }

/\*

Challenge:

  1. Create an array of promises using getImagePromise.

  2. Save the results of calling all of those promises

     in one go to a const 'results'.

  3. If the promises resolve:

     - log "All images loaded successfully!".

     - hide 'uploadContainer'

     - iterate over the results and render them to imgContainer.

  4. If the promises reject:

     - catch and log the error.

\*/

}

document.getElementById('submit-imgs').addEventListener('click', () => preloadImages(images))

------------------

const books = {

    "b001": { title: "To Kill a Mockingbird", price: 18.99, isAvailable: true },

    "b002": { title: "1984", price: 15.99, isAvailable: false },

    "b003": { title: "The Great Gatsby", price: 12.49, isAvailable: true },

    "b004": { title: "Moby Dick", price: 22.50, isAvailable: false }

  }

  /\*

  Challenge:

    1. Create an array of values using Object.values

    2. Iterate over it and log out each book price.

  \*/

  const bookValues = Object.values(books)

  bookValues.forEach(val => console.log(val.price))

  --------------------------------

  ##Object Method Challenge

  const books = {

    "b001": { title: "To Kill a Mockingbird", price: 18.99, isAvailable: true },

    "b002": { title: "1984", price: 15.99, isAvailable: false },

    "b003": { title: "The Great Gatsby", price: 12.49, isAvailable: true },

    "b004": { title: "Moby Dick", price: 22.50, isAvailable: false }

  }

  /\*

  Challenge:

    1. Create an array of values using Object.values

    2. Iterate over it and log out each book price.

  \*/

  const bookValues = Object.values(books)

  bookValues.forEach(val => console.log(val.price))

function Character(name){

    this.name = name

    this.collectedItemsArr = []

    this.addItem = function(item) {

        this.collectedItemsArr.push(item)

        console.log(`${this.name} now has: ${this.collectedItemsArr.join(',')}`)

    }

}

const wizard = new Character('Merlin')

const witch = new Character('Hermione')

wizard.addItem('wand')

witch.addItem('map')

//Simple Generator

function\* simpleGenerator() {

    yield 1; // Pauses here

    yield 2; // Pauses here

    yield 3; // Pauses here

}

const gen = simpleGenerator();

console.log(gen.next()); // { value: 1, done: false }

console.log(gen.next()); // { value: 2, done: false }

console.log(gen.next()); // { value: 3, done: false }

console.log(gen.next()); // { value: undefined, done: true }

//Infinite Sequence Generator

function\* infiniteSequence() {

    let i = 0;

    while (true) {

        yield i++;

    }

}

const seq = infiniteSequence();

console.log(seq.next().value); // 0

console.log(seq.next().value); // 1

console.log(seq.next().value); // 2

// Iterating with for ..... of

function\* colorGenerator() {

    yield 'Red';

    yield 'Green';

    yield 'Blue';

}

for (const color of colorGenerator()) {

    console.log(color);

}

// Output:

// Red

// Green

// Blue

// Simulating Async Execution with Generators

function\* asyncTask() {

    console.log('Start Task');

    const result1 = yield new Promise(resolve => setTimeout(() => resolve('Step 1 Complete'), 1000));

    console.log(result1);

    const result2 = yield new Promise(resolve => setTimeout(() => resolve('Step 2 Complete'), 1000));

    console.log(result2);

    console.log('End Task');

}

// Custom runner function to handle async flow

function run(generator) {

    const iterator = generator();

    function handleNext(promise) {

        if (promise.done) return;

        promise.value.then(result => handleNext(iterator.next(result)));

    }

    handleNext(iterator.next());

}

run(asyncTask);

// Output:

// Start Task

// Step 1 Complete

// Step 2 Complete

// End Task

//Debouncing and Generator in JS

//DataAPI JS

export const cafeDataArr = [

    {

        heading: 'Our Commitment to Our Growers',

        teaser: 'Our growers enjoy the immense benefits of our fair trade practices and sustainable farming initiatives.'

    },

    {

        heading: 'Explore Our Menu',

        teaser: 'Discover our selection of handcrafted beverages and gourmet snacks tailored to tantalize your taste buds.'

    },

    {

        heading: 'Brewing Techniques',

        teaser: 'Learn the art of coffee making with our guide to various brewing methods from around the world.'

    },

    {

        heading: 'Coffee House Events',

        teaser: 'Join us for live music, poetry readings, and other exciting events at our local coffee house.'

    },

    {

        heading: 'Our Sustainability Journey',

        teaser: 'We are committed to reducing our environmental footprint, one cup at a time. Discover our eco-friendly initiatives.'

    },

    {

        heading: 'Meet Our Baristas',

        teaser: 'Get to know the passionate professionals who craft your perfect cup of coffee every day.'

    },

    {

        heading: 'Membership Perks',

        teaser: 'Become a member to enjoy exclusive discounts, special offers, and early access to our new products.'

    },

    {

        heading: 'Coffee Origins',

        teaser: 'Trace the journey of our beans from lush plantations across the globe to your coffee cup.'

    },

    {

        heading: 'Gift Cards & Merchandise',

        teaser: 'Find the perfect gift for the coffee lover in your life with our range of gift cards and branded merchandise.'

    },

    {

        heading: 'Visit Us',

        teaser: 'Experience the warmth and aroma of our coffee house by visiting us at one of our cozy locations.'

    }

]

// End DataAPI JS

//Index.js

/\*

When the page loads, this site should display the

'header', 'hero', and 'nav' sections only.

As a user scrolls, more content should be added.

There are ten content sections in total, which are

loading from apiData.js

The handleScroll function should be debounced to

limit the number of times a scroll event triggers it.

If no debounce time is passed in, it should default

to 100 milliseconds.

At the moment, the code is broken.

Challenge:

1. Identify and fix the bugs in this code.

\*/

import { cafeDataArr } from '/apiData.js'

function\* fetchDataGenerator(maxSections = 10) {

    let sectionCount = 0

    while (true) {

        sectionCount++

        if (sectionCount >= maxSections) {

            console.log("Max section limit reached, stopping generator.")

            return

        }

        const fakeApiResponse = { sectionText: cafeDataArr[sectionCount] }

        // Simulate an asynchronous API call with a promise

        yield new Promise(resolve => setTimeout(() => resolve(fakeApiResponse), 100))

    }

}

const generator = fetchDataGenerator()

function handleScroll() {

    const result = generator.next()

    console.log(result)

    if (!result.done) {

        result.value.then(data => {

            // Process and display the data

            const contentSection = document.createElement('section')

            const sectionHeader = document.createElement('h3')

            const sectionTeaser = document.createElement('p')

            sectionHeader.innerText = data.sectionText.heading

            sectionTeaser.innerText = data.sectionText.teaser

            contentSection.appendChild(sectionHeader)

            contentSection.appendChild(sectionTeaser)

            document.body.appendChild(contentSection)

        }).catch(error => {

            console.error('Failed to load section:', error)

        })

    } else {

        console.log('No more sections to load.')

    }

}

// Debouncing function

function debounce(func, timeout = 100) {

    let timer

    return function () {

        clearTimeout(timer)

        timer = setTimeout(() => {

            func()

        }, timeout)

    }

}

// Attach debounced handler to scroll event

document.addEventListener('scroll', debounce(handleScroll, 100))

res.data.products = ['Product 1', 'Product 2', 'Product 3'];

**Output HTML for fetch products from the server:-**

<select id="selProd" onChange="getDealers()">

<option value="--SELECT--">--SELECT--</option>

<option value="Product 1">Product 1</option>

<option value="Product 2">Product 2</option>

<option value="Product 3">Product 3</option>

</select>

<html>

  <script src="https://cdn.jsdelivr.net/npm/axios/dist/axios.min.js"></script>

  <style>

    td:nth-child(even), th:nth-child(even) {

      background-color: #D6EEEE;

    }

  </style>

  <script>

    let produrl = "http://localhost:5000/";

    let dealerurl = "https://dealerdetails.1qifx92m49y2.us-south.codeengine.appdomain.cloud";

    // Fetch products from the server

    axios.get(produrl + "products").then(res => {

      let selEl = document.createElement("select");

      selEl.setAttribute("id", "selProd");

      selEl.setAttribute("onChange", "getDealers()");

      let defaultOpt = document.createElement("option");

      defaultOpt.setAttribute('value', "--SELECT--");

      defaultOpt.innerText = "--SELECT--";

      selEl.appendChild(defaultOpt);

      // Add products to the dropdown

      res.data.products.forEach(element => {

        let optEl = document.createElement("option");

        optEl.setAttribute('value', element);

        optEl.innerText = element;

        selEl.appendChild(optEl);

      });

      // Insert the product dropdown into the page

      let prod = document.getElementById('prod\_dd');

      prod.childNodes.forEach(node => prod.removeChild(node));

      prod.appendChild(selEl);

    });

    // Fetch dealers for the selected product

    let getDealers = () => {

      let valToFetch = document.getElementById("selProd").value;

      if (valToFetch != "--SELECT--") {

        axios.get(produrl + "getdealers/" + valToFetch).then(res => {

          let selEl = document.createElement("select");

          selEl.setAttribute("id", "selDealer");

          selEl.setAttribute("onChange", "getDealerProductPrice()");

          let defaultOpt = document.createElement("option");

          defaultOpt.setAttribute('value', "--SELECT--");

          defaultOpt.innerText = "--SELECT--";

          selEl.appendChild(defaultOpt);

          // Add dealers to the dropdown

          res.data.dealers.forEach((element) => {

            let optEl = document.createElement("option");

            optEl.setAttribute('value', element);

            optEl.innerText = element;

            selEl.appendChild(optEl);

          });

          // Add 'All Dealers' option

          let optEl = document.createElement("option");

          optEl.setAttribute('value', 'All Dealers');

          optEl.innerText = 'All Dealers';

          selEl.appendChild(optEl);

          // Insert the dealer dropdown into the page

          let dealers = document.getElementById('dealers');

          dealers.childNodes.forEach(node => dealers.removeChild(node));

          dealers.appendChild(selEl);

        });

      }

    }

    // Fetch price for the selected dealer and product

    let getDealerProductPrice = () => {

      let prodvalToFetch = document.getElementById("selProd").value;

      let dealervalToFetch = document.getElementById("selDealer").value;

      let pricingEl = document.getElementById("pricing");

      pricingEl.childNodes.forEach(node => pricingEl.removeChild(node));

      // Fetch price for specific dealer

      if (dealervalToFetch != 'All Dealers') {

        axios.get(dealerurl + "price/" + dealervalToFetch + "/" + prodvalToFetch).then(res => {

          document.getElementById("pricing").innerText = res.data.message;

        });

      } else {

        // Fetch prices for all dealers

        axios.get(dealerurl + "allprice/" + prodvalToFetch).then(res => {

          let table = document.createElement("table");

          res.data.prices.forEach((price) => {

            let tr = document.createElement("tr");

            tr.innerHTML = "<td>" + price.key + "</td><td>" + price.value + "</td>";

            table.appendChild(tr);

          });

          // Insert table into the page

          if (table.hasChildNodes()) {

            pricingEl.appendChild(table);

          }

        });

      }

    }

  </script>

  <h1>

    Products price comparison

    <div style="margin:6mm" id="prod\_dd"></div>

    <div style="margin:6mm" id="dealers"></div>

    <div style="margin:6mm;font-size: large;" id="pricing"></div>

  </h1>

</html>

import React, { useState, useEffect } from 'react';

import axios from 'axios';

const PriceComparison = () => {

  const [products, setProducts] = useState([]);

  const [dealers, setDealers] = useState([]);

  const [prices, setPrices] = useState([]);

  const [selectedProduct, setSelectedProduct] = useState('');

  const [selectedDealer, setSelectedDealer] = useState('');

  const [loading, setLoading] = useState(false);

  const produrl = "http://localhost:5000/";

  const dealerurl = "https://dealerdetails.1qifx92m49y2.us-south.codeengine.appdomain.cloud";

  // Fetch products on component mount

  useEffect(() => {

    axios.get(produrl + "products")

      .then(res => setProducts(res.data.products))

      .catch(error => console.error("Error fetching products:", error));

  }, []);

  // Fetch dealers when a product is selected

  useEffect(() => {

    if (selectedProduct && selectedProduct !== '--SELECT--') {

      axios.get(produrl + "getdealers/" + selectedProduct)

        .then(res => setDealers(res.data.dealers))

        .catch(error => console.error("Error fetching dealers:", error));

    }

  }, [selectedProduct]);

  // Fetch price when a dealer is selected or for all dealers

  const getDealerProductPrice = () => {

    setLoading(true);

    if (selectedDealer !== 'All Dealers') {

      axios.get(dealerurl + "price/" + selectedDealer + "/" + selectedProduct)

        .then(res => {

          setPrices([res.data.message]);

          setLoading(false);

        })

        .catch(error => {

          console.error("Error fetching price:", error);

          setLoading(false);

        });

    } else {

      axios.get(dealerurl + "allprice/" + selectedProduct)

        .then(res => {

          setPrices(res.data.prices);

          setLoading(false);

        })

        .catch(error => {

          console.error("Error fetching all prices:", error);

          setLoading(false);

        });

    }

  };

  return (

    <div>

      <h1>Products Price Comparison</h1>

      <div style={{ margin: '6mm' }}>

        <select

          id="selProd"

          value={selectedProduct}

          onChange={(e) => setSelectedProduct(e.target.value)}

        >

          <option value="--SELECT--">--SELECT--</option>

          {products.map((product, index) => (

            <option key={index} value={product}>

              {product}

            </option>

          ))}

        </select>

      </div>

      {selectedProduct !== '--SELECT--' && (

        <div style={{ margin: '6mm' }}>

          <select

            id="selDealer"

            value={selectedDealer}

            onChange={(e) => setSelectedDealer(e.target.value)}

          >

            <option value="--SELECT--">--SELECT--</option>

            {dealers.map((dealer, index) => (

              <option key={index} value={dealer}>

                {dealer}

              </option>

            ))}

            <option value="All Dealers">All Dealers</option>

          </select>

        </div>

      )}

      <div style={{ margin: '6mm', fontSize: 'large' }}>

        {loading ? (

          <p>Loading...</p>

        ) : (

          prices.length > 0 && (

            <div>

              {selectedDealer !== 'All Dealers' ? (

                <p>{prices[0]}</p>

              ) : (

                <table>

                  <tbody>

                    {prices.map((price, index) => (

                      <tr key={index}>

                        <td>{price.key}</td>

                        <td>{price.value}</td>

                      </tr>

                    ))}

                  </tbody>

                </table>

              )}

            </div>

          )

        )}

      </div>

    </div>

  );

};

export default PriceComparison;