**Prepare**

Here's an example of a JavaScript object representing a person:

**let person = {**

**name: "Antonia Francesca",**

**age: 30,**

**profession: "Software Engineer",**

**hobbies: ["reading", "playing guitar", "hiking"],**

**address: {**

**street: "123 Camino Real",**

**city: "Santa Rosa",**

**country: "Honduras"**

**},**

**isEmployed: true,**

**greet: function() {**

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

**}**

**};**

In this example, the person object has various properties such as name, age, profession, hobbies, address, and isEmployed. It also includes a greet method that logs a greeting message to the console using the person's name.

The address property is an object itself, containing nested properties such as street, city, and country.

You can access and modify the properties and invoke the methods of the person object using dot notation or bracket notation. For example:

**console.log(person.hobbies[0]); // Output: reading**

## Array Methods for Functional Programming

### Overview

Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions or declarations instead of statements. Array methods that support functional programming include **filter()**, **map()**, and **reduce()**. There are other array methods that support functional programming, but this learning activity dives into these three common and useful methods.

### Prepare

#### **array.filter()**

The array.filter() method creates a filtered array from the original array using the condition from the provided function.

#### **array.map()**

The array.map() method iterates through each element of the original array using a given function and produces a new array. The original array is not modified. The new array is returned by the map() method. The new array will have the same number of elements as the original array.

#### **array.reduce()**

The array.reduce() method is used to reduce the array to a single value. It executes a reducer function on each element of the array, resulting in a single output value. The reducer function takes four arguments: Accumulator, Current Value, Current Index, Source Array. The reducer function's returned value is assigned to the accumulator, whose value is remembered across each iteration throughout the array and ultimately becomes the final, single resulting value.

### Check Your Understanding

For all of these exercises, use this array declaration:

**let** names = ['Nancy','Blessing','Jorge','Svetlana','Bob'];

* Using the ***array.filter()*** method, create an array named **namesB** with **only** those names from the **name** array that start with the character 'B'.Answer

**let** namesB = names.**filter**(name => name.**charAt**(0) === 'B');

* Using the ***array.map()*** method, create a new array named **namesLength** that contains the length of each name in the **names** array. Expected output is **[5, 8, 5, 8, 3]**.Answer

**let** nameLengths = names.**map**((name) => name.length);

* Using the ***names.reduce()*** method, create an expression that returns the **average string length** of the names in the **names** array. Expected output is **5.8**.Answer

names.**reduce**((total, name) => total + name.length, 0) / names.length;

The initial value of the reduction is set to 0. Inside the reducer function, total represents the accumulated sum of string lengths, and name represents each individual name in the array. The reducer function adds the length of each name to the total in each iteration. (like any accumulator, total += name.length).