What is the difference between instance methods and static methods?

Instance Methods vs Static Methods:

Instance Methods basically belongs to the Object of a class whereas Static Methods belongs to the class itself.

Instance Method Example:

class Car {

  constructor(brand, model) {

    this.brand = brand;

    this.model = model;

  }

  // Instance method

  displayInfo() {

    console.log(`This car is a ${this.brand} ${this.model}.`);

  }

}

const myCar = new Car('Toyota', 'Corolla');

myCar.displayInfo(); // This car is a Toyota Corolla.

Static Method Example:

class Car {

  constructor(brand, model) {

    this.brand = brand;

    this.model = model;

  }

  // Instance method

  displayInfo() {

    console.log(`This car is a ${this.brand} ${this.model}.`);

  }

  // Static method

  static compareCars(car1, car2) {

    return car1.model === car2.model && car1.brand === car2.brand;

  }

}

const car1 = new Car('Toyota', 'Corolla');

const car2 = new Car('Toyota', 'Camry');

console.log(Car.compareCars(car1, car2)); // false

**How javaScript Handles concurrency?**

JavaScript handles concurrency by using the Web API, Callback Queue and Event Loop

Example:

console.log('Start'); // printed first

setTimeout(() => {

  console.log('Timeout callback'); // printed last

}, 1000);

console.log('End'); //printed second

Execution Flow:

1. Global Execution Context (GEC) is created and pushed into the callstack.
2. “start” is printed
3. SetTimeout() is called and the callback gets registered with the Web API and a timer gets set to 1000ms.
4. “End” is printed
5. After 1000ms the callback of setTimeout() is moved to Callback Queue
6. The Event Loop checks the Call Stack if it is empty. If it is empty, then it removes the setTimeout() callback from the Callback Queue and pushes it to the Call Stack.
7. “Timeout Callback” gets printed

**What is async/await? How does it differ from using the promise instance methods?**

Async and Await are keywords in JavaScript that makes Asynchronous code behave like Synchronous code.

Declaring a function as Async automatically makes it to return a promise. Inside Async function we need to use Await keyword to perform Asynchronous operations e.g. fetch. The Await keywords pauses the execution flow until the asynchronous operation is completed and the promise is resolved or rejected.

Example:

async function fetchData() {

  try {

    const response = await fetch('https://api.example.com/data');

    const data = await response.json();

    console.log(data);

  } catch (error) {

    console.error('Error fetching data:', error);

  }

}

fetchData();

Async/await differs from promise instance methods (then, catch, finally) in terms of syntax.

Example :

fetch('https://api.example.com/data')

  .then(response => response.json())

  .then(data => {

    const processedData = processData(data);

    console.log(processedData);

  })

  .catch(error => {

    console.error('Error:', error);

  });

**Can you use await outside of an async function?**

No, we can’t use await outside of an async function. Await must be used inside async function.

**What is callback hell and why is it considered a problem?**

A callback hell is considered as a situation in which multiple nested callbacks are involved in an asynchronous operation.

It is considered as a problem because it makes the code difficult to read and debug.