**Practical 5**

**AIM:**

Define a class representing a vehicle with properties like make, model, and year. Implement methods to display the vehicle details and calculate the mileage.

Create child classes like Car and Motorcycle that inherit from the Vehicle class and add specific properties and methods.

CODE:

class Vehicle {

  constructor(make, model, year) {

    this.make = make;

    this.model = model;

    this.year = year;

  }

  displayDetails() {

    console.log(`Vehicle: ${this.year} ${this.make} ${this.model}`);

  }

  calculateMileage(distance, fuel) {

    const mileage = distance / fuel;

    console.log(`Mileage: ${mileage} miles per liter`);

  }

}

class Motorcycle extends Vehicle {

  constructor(make, model, year, engineDisplacement) {

    super(make, model, year);

    this.engineDisplacement = engineDisplacement;

  }

  displayDetails() {

    super.displayDetails();

    console.log(`Engine Displacement: ${this.engineDisplacement} cc`);

  }

}

class Car extends Vehicle {

  constructor(make, model, year, doors) {

    super(make, model, year);

    this.doors = doors;

  }

  displayDetails() {

    super.displayDetails();

    console.log(`Doors: ${this.doors}`);

  }

}

const motorcycle = new Motorcycle("Kawasaki", 2023, 400);

motorcycle.displayDetails();

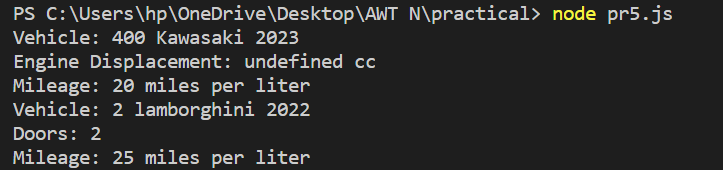
motorcycle.calculateMileage(200, 10);

const car = new Car("lamborghini", 2022, 2, 2);

car.displayDetails();

car.calculateMileage(350, 14);

OUTPUT:



**Practical 6**

**AIM:**

Use the prototype property to add a new method to an existing object constructor, such as Array or String.

**CODE:**

Array.prototype.add = function () {

  let total = 0;

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

    total += this[i];

  }

  return total;

};

const numbers = [15, 30, 45];

console.log(numbers.add());

**OUTPUT:**



**Practical 7**

**AIM:**

Create a JavaScript module that exports a class representing a calculator with methods to perform basic arithmetic operations. Import the module in another JavaScript file and use the calculator class to perform calculations.

**CODE:**

**Pr7.js**

const Cal = require("./export");

const cal = new Cal();

const Add = cal.add(10, 5);

console.log("10 + 20 =", Add);

const Subtract = cal.subtract(10, 5);

console.log("10 - 20 =", Subtract);

const Multiply = cal.multiply(10, 5);

console.log("10 \* 20 =", Multiply);

const Divide = cal.divide(10, 5);

console.log("10 / 20 =", Divide);

**export.js**

class Calculator {

  add(a, b) {

    return a + b;

  }

  subtract(a, b) {

    return a - b;

  }

  multiply(a, b) {

    return a \* b;

  }

  divide(a, b) {

    if (b === 0) {

      throw new Error("Cannot divide by zero");

    }

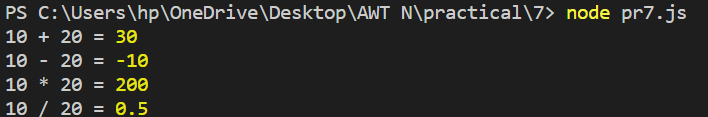
    return a / b;

  }

}

module.exports = Calculator;

**OUTPUT:**



**Practical 8**

**AIM:**

Create a JavaScript module that fetches data from an API using the fetch() function and exports the retrieved data.

Create an async function getUsers(names), that gets an array of GitHub logins, fetches the users from GitHub and returns an array of GitHub users.

The GitHub url with user information for the given USERNAME is: https://api.github.com/users/USERNAME.

There’s a test example in the sandbox.

Important details:

* There should be one fetch request per user.
* Requests shouldn’t wait for each other. So that the data arrives as soon as possible.
* If any request fails, or if there’s no such user, the function should return null in the resulting array.

**CODE:**

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8" />

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

    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/bootswatch/4.5.3/lux/bootst

rap.min.css" />

    <title>Github Finder</title>

</head>

<body>

    <nav class="navbar navbar-dark bg-primary mb-3">

        <div class="container">

            <a href="#" class="navbar-brand">Github Finder</a>

        </div>

    </nav>

    <div class="container searchContainer">

        <div class="search card card-body">

            <h1 class="text-warning">Search Github Users</h1>

            <p class="lead">Enter a username to fetch Github profiles and

                repos</p>

            <input type="text" class="form-control" id="searchUser" placeholder="Github Username..." />

        </div>

        <br />

        <div id="profile"></div>

    </div>

    <footer class="mt-5 p-3 text-center bg-light">Github Finder

        &copy;</footer>

    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-

DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrCXaRkfj" crossorigin="anonymous"></script>

    <script src="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/js/bootstrap.bund

le.min.js" integrity="sha384-

ho+j7jyWK8fNQe+A12Hb8AhRq26LrZ/JpcUGGOn+Y7RsweNrtN/tE3MoK7ZeZDyx" crossorigin="anonymous"></script>

    <script src="github.js"></script>

    <script src="ui.js"></script>

    <script src="app.js"></script>

</body>

</html>

**app.js**

// Init/Instantiate

const github = new Github();

const ui = new UI();

// Search input

const searchUser = document.getElementById("searchUser");

// Search input event listener

searchUser.addEventListener("keyup", (e) => {

  // Get input text

  const userText = e.target.value;

  if (userText !== "") {

    // console.log(userText);

    // Make http

    github.getUser(userText).then((data) => {

      // console.log(data);

      if (data.profile.message === "Not Found") {

        // Show alert

        ui.showAlert("User not found", "alert alert-danger");

      } else {

        // Show profile

        ui.showProfile(data.profile);

        ui.showRepos(data.repos);

      }

    });

  } else {

    // Clear profile

    ui.clearProfile();

  }

});

**github.js**

class Github {

  constructor() {

    this.client\_id = "cf0d333f57abe0450b4c";

    this.client\_secret = "3b4bf04f61638b6a8dd3a9cd42173c2049eea529";

    this.repos\_count = 5;

    this.repos\_sort = "create: asc";

  }

  async getUser(user) {

    const profileResponse = await fetch(

      `https://api.github.com/users/${user}?client\_id=${this.client\_id}&client

\_secret=${this.client\_secret}`

    );

    const repoResponse = await fetch(

      `https://api.github.com/users/${user}/repos?per\_page=${this.repos\_count}

&sort=${this.repos\_sort}&client\_id=${this.client\_id}&client\_secret=${this.client\_secret}`

    );

    const profile = await profileResponse.json();

    const repos = await repoResponse.json();

    return {

      profile,

      repos,

    };

  }

}

**ui.js**

class UI {

  constructor() {

    this.profile = document.getElementById("profile");

  }

  showProfile(user) {

    // console.log(user);

    const date = new Date(user.created\_at);

    this.profile.innerHTML = `

<div class="card card-body mb-3">

<div class="row">

<div class="col-md-3">

<img class="img-fluid mb-2" src="${user.avatar\_url}">

<a href="${

      user.html\_url

    }" target="\_blank" class="btn btn-primary btn-block">View

Profile</a>

</div>

<div class="col-md-9">

<span class="badge badge-info">Public Repos: ${user.public\_repos}</span>

<span class="badge badge-info">Public Gists: ${user.public\_gists}</span>

<span class="badge badge-info">Followers: ${user.followers}</span>

<span class="badge badge-info">Following: ${user.following}</span>

<br><br>

<ul class="list-group">

<li class="list-group-item">Company: ${user.company}</li>

<li class="list-group-item">Blog: ${user.blog}</li>

<li class="list-group-item">Location: ${user.location}</li>

<li class="list-group-item">Member Since:

${date.toLocaleDateString()}</li>

</ul>

</div>

</div>

</div>

<h3 class="page-heading mb-3">Latest Repos</h3>

<div. id="repos"></div>

`;

  }

  // Show user repos

  showRepos(repos) {

    let output = "";

    repos.forEach((repo) => {

      output += `

<div class="card card-body mb-2>

<div class ="row">

<div class="col-md-6">

<a href="${repo.html\_url}" target="\_blank">${repo.name}</a>

</div>

<div class="col-md-6">

<span class="badge badge-primary">Public Stars:

${repo.stargazers\_count}</span>

<span class="badge badge-info">Public Watchers:

${repo.watchers\_count}</span>

<span class="badge badge-warning">Public Forks:

${repo.forks\_count}</span>

<span class="badge badge-success">Primary Language:

${repo.language}</span>

</div>

</div>

</div>

`;

    });

    // Output repos

    document.getElementById("repos").innerHTML = output;

  }

  // Show Alert Message

  showAlert(message, className) {

    // Clear any remaining alerts

    this.clearAlert();

    // Create div

    const div = document.createElement("div");

    // Add classes

    div.className = className;

    // Add text

    div.appendChild(document.createTextNode(message));

    // Get parent, Get search box, Insert before search box

    const container = document.querySelector(".searchContainer");

    const search = document.querySelector(".search");

    container.insertBefore(div, search);

    // timeout after 3 seconds

    setTimeout(() => {

      this.clearAlert();

    }, 3000);

  }

  // Clear alert

  clearAlert() {

    const currentAlert = document.querySelector(".alert");

    if (currentAlert) {

      currentAlert.remove();

    }

  }

  // Clear profile

  clearProfile() {

    this.profile.innerHTML = "";

  }

}

**Practical 9**

**AIM:**

Implement dynamic imports using the import() function to load modules asynchronously based on certain conditions

**CODE:**

**Pr9.js**

const condition = true;

if (condition) {

  import("./module1.mjs")

    .then((module1) => {

      module1.Hello1();

    })

    .catch((error) => console.error("Error importing Module A:", error));

} else {

  import("./module2.mjs")

    .then((module2) => {

      module2.Hello2();

    })

    .catch((error) => console.error("Error importing Module B:", error));

}

**module1.js**

export function Hello1() {

  console.log("Hello from Module 1!");

}

**module2.js**

export function Hello1() {

  console.log("Hello from Module 2!");

}

**OUTPUT:**



**Practical 10**

**AIM:**

Create an iterator that generates an infinite sequence of numbers and a generator that yields a sequence of even numbers. Use the iterator and generator in different scenarios.

**CODE:**

function\* infinite() {

  let i = 0;

  while (true) {

    yield i++;

  }

}

function\* evenNum(count) {

  let i = 0;

  let iteratedCount = 0;

  while (iteratedCount < count) {

    if (i % 2 === 0) {

      yield i;

      iteratedCount++;

    }

    i++;

  }

}

const Iterator = infinite();

for (let i = 0; i < 5; i++) {

  console.log("Infinite sequence:", Iterator.next().value);

}

const evenNumGenerator = evenNum(5);

for (const evenNumber of evenNumGenerator) {

  console.log("Even number:", evenNumber);

}

**OUTPUT:**

