Assignment: More on JavaScript

Q1: Understanding Arrow Functions & Lexical this

Consider the following code and **predict the output**. Explain why the behavior occurs.

```
Unset
const person = {
    name: "Alice",
    greet: function() {
        setTimeout(() => {
            console.log(`Hello, my name is ${this.name}`);
        }, 1000);
    }
};

person.greet();
```

Ans) **JS code**: link

Inside the arrow function `this` is resolved lexically i.e. it first checks if `this` is available in the arrow function, here it is not, then it checks inside the object method `greet` where `this` refers to the object `person` itself where `this` is defined and `this.name` refers to `person.name`.

Q2: Convert Traditional Functions to Arrow Functions

Rewrite the following function using **arrow functions** without changing the behavior:

Original Code:

```
Unset
function multiply(a, b) {
    return a * b;
}

const obj = {
    value: 10,
    add: function(num) {

        return this.value + num;
    }
};

console.log(multiply(5, 3));
console.log(obj.add(5));
```

Ans) **JS code** : link

Q3: Handling Errors in Async-Await (Try-Catch Required)

Modify the following function to **use async-await and proper error handling (`try-catch`).**

Original Code (With Promise):

```
Unset
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            let success = Math.random() > 0.5;
            success ? resolve("Data received") : reject("Error
fetching data");
        }, 2000);
    });
}

fetchData().then(console.log).catch(console.error);
```

Ans) **JS code** : link

Q4: Async Function with Multiple Await Calls

Write an async function that:

1. Fetches **user data** after 1 second.
2. Fetches **order details** after 2 seconds.
3. Fetches **payment status** after 3 seconds.
4. Logs the final **"Order completed"** message.
5. Use `await` to ensure **each step executes sequentially.**

Ans) **JS code** : link

Q5: Handling Synchronous Errors with `try-catch`

Write a function that:

1. Accepts a string input.
2. Converts it to a number and returns its square.
3. Uses `try-catch` to handle cases where the input is not a valid number.
4. If the error occurs, return `"Invalid Input"` instead of crashing.

Ans) **JS code** : link