JAVASCRIPT AND TYPESCRIPT

JavaScript and TypeScript are two widely-used programming languages, particularly in web development. Both have their own strengths, applications, and key differences. Below is an overview of each, followed by example codes and a comparison.

JavaScript

JavaScript is a high-level, interpreted programming language that is primarily used to make web pages interactive. It is a core technology of the web, alongside HTML and CSS. JavaScript is dynamic, weakly typed, and supports multiple programming paradigms, including object-oriented, functional, and event-driven programming.

Example Code (JavaScript)

```
// Function to calculate the square of a number

function square(number) {

  return number * number;

}


// Calling the function

console.log(square(5)); // Output: 25


// Using an object and a method

const person = {

  firstName: 'John',

  lastName: 'Doe',

  getFullName: function() {

    return `${this.firstName} ${this.lastName}`;

  }
};

console.log(person.getFullName()); // Output: John Doe
```

TypeScript

TypeScript is a superset of JavaScript developed by Microsoft. It adds static typing, interfaces, and other features that enhance the development experience, particularly in large projects. TypeScript code is transpiled into JavaScript, ensuring compatibility with browsers and environments that support JavaScript.

Key Features of TypeScript

- Static typing
- Interfaces and enums
- Better tooling and IDE support
- Supports all JavaScript features

Example Code (TypeScript)

```
// Defining a typed function
function square(number: number): number {
   return number * number;
}

// Calling the function
console.log(square(5)); // Output: 25

// Using an interface and a class
interface Person {
   firstName: string;
```

```
  lastName: string;
  getFullName(): string;
}

class Employee implements Person {
  firstName: string;
  lastName: string;

  constructor(firstName: string, lastName: string) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
}

const employee = new Employee('Jane', 'Doe');
console.log(employee.getFullName()); // Output: Jane Doe
```

Similarities between JavaScript and TypeScript

1. **Syntax**: TypeScript extends JavaScript syntax, so valid JavaScript code is also valid TypeScript code.
2. **Runtime Environment**: Both run in JavaScript environments (e.g., browsers, Node.js).

3. **Use Cases**: Both are used for web development, server-side programming, and building applications.
4. **Event Handling**: Both support event-driven programming.

Differences between JavaScript and TypeScript

Feature	**JavaScript**	**TypeScript**
Typing	Dynamically typed	Statically typed
Compilation	Interpreted at runtime	Transpiled to JavaScript
IDE Support	Basic support	Enhanced with static type checking
Learning Curve	Easier to learn for beginners	Slightly steeper due to additional features
Error Detection	Runtime errors	Compile-time error detection
Features	Basic ECMAScript features	Advanced features like interfaces, enums

---

Conclusion

Both JavaScript and TypeScript are powerful tools for web development. JavaScript is ideal for quick projects and scripts, while TypeScript shines in larger applications where maintainability and scalability are crucial. Understanding their similarities and differences helps developers choose the right tool for their specific needs.

JavaScript and TypeScript can support both object-oriented and structured programming paradigms, but they are typically used as object-oriented languages in modern development. Here's an overview of object-oriented programming **Fundamentals to Learn**

1. **Objects and Classes**:

    - Understand how objects are defined and used to model real-world entities.
    - Learn about classes
2. **Encapsulation**:
    - Grasp how data can be hidden within objects, using access modifiers (public, private, protected) in TypeScript.
3. **Inheritance**:
    - Learn how one class can inherit properties and methods from another, using extends in both JavaScript and TypeScript.
4. **Polymorphism**:
    - Understand how methods can be overridden or how different objects can define the same method differently.
5. **Abstraction**:
    - Focus on hiding complexity and exposing only what is necessary, using interfaces or abstract classes (available in TypeScript but not directly in JavaScript).
6. **Composition**:
    - Learn to build complex objects by combining simpler objects rather than relying solely on inheritance.

Are JavaScript and TypeScript Object-Oriented or Structured?

- **JavaScript**:
    - JavaScript is a multi-paradigm language that supports object-oriented, functional, and procedural programming. While it was initially more structured, modern JavaScript heavily adopts object-oriented principles, especially with the introduction of classes in ES6.
- **TypeScript**:
    - TypeScript extends JavaScript's OOP capabilities by adding static typing and features like interfaces, enums, and abstract classes. These

additions make TypeScript more suitable for traditional OOP practices.

Codes for calculating tip and bill in Javascript and in Typescript.

Quiz: If bill is 300 sh plus, the tip is 5% and if the bill is between 50 and 200, the tip is 3%. Now calculate tip and amount paid if the bill is 250, 500 and 700.

JAVASCRIPT

```
// Calculate tip and amount paid based on bill value
function calculateTip(bill) {
  let tip = 0;
  if (bill > 300) {
    tip = bill * 0.05;
  } else if (bill >= 50 && bill <= 200) {
    tip = bill * 0.03;
  }
  const total = bill + tip;
  return { tip, total };
}

// Bills
const bills = [250, 500, 700];

bills.forEach(bill => {
  const { tip, total } = calculateTip(bill);
  console.log(`Bill: ${bill}, Tip: ${tip.toFixed(2)}, Total: ${total.toFixed(2)}`);
});
```

TYPESCRIPT

```
// Calculate tip and amount paid based on bill value
function calculateTip(bill: number): { tip: number; total: number } {
  let tip = 0;
  if (bill > 300) {
    tip = bill * 0.05;
  } else if (bill >= 50 && bill <= 200) {
    tip = bill * 0.03;
  }
  const total = bill + tip;
  return { tip, total };
}

// Bills
const bills: number[] = [250, 500, 700];

bills.forEach(bill => {
  const { tip, total } = calculateTip(bill);
  console.log(`Bill: ${bill}, Tip: ${tip.toFixed(2)}, Total: ${total.toFixed(2)}`);
});
```

Key Differences in the Code:

1. **Static Typing**:
   - **JavaScript**: Variables and parameters are dynamically typed. There are no explicit type annotations for the bill or return values of functions.

JAVASCRIPT

```
function calculateTip(bill) {
    let tip = 0;
    if (bill > 300) {
        tip = bill * 0.05;
    } else if (bill >= 50 && bill <= 200) {
        tip = bill * 0.03;
    }
    const total = bill + tip;
    return { tip, total };
}
```

TYPESCRIPT: 1. Variables and parameters have explicitly defined types, ensuring compile-time type checking.

```
function calculateTip(bill: number): { tip: number; total: number } {
    let tip = 0;
    if (bill > 300) {
        tip = bill * 0.05;
    } else if (bill >= 50 && bill <= 200) {
        tip = bill * 0.03;
    }
    const total = bill + tip;
    return { tip, total };
}
```

2. **Error Detection**:
    - **JavaScript**: Errors related to data types are only detected at runtime.

   - **TypeScript**: Errors such as passing non-numeric values to the bill parameter are detected at compile time.

3. **Object Type Annotations**:
   - In TypeScript, the return value of the calculateTip function is annotated as { tip: number; total: number }, making it clear what structure is expected. JavaScript does not provide such declarations.

4. **Array Typing**:
   - **JavaScript**: Arrays can hold any type of data without restrictions.

javascript

```
const bills = [250, 500, 700];
```

   - **TypeScript**: Arrays have explicit types, ensuring they only contain numbers.

typescript

```
const bills: number[] = [250, 500, 700];
```