Features of JavaScript

Core features of java script

1. Dynamically Typed Language

- In JavaScript, there is no need to explicitly declare the data type of a variable.
- The type is determined at runtime based on the value assigned to the variable.

```
let num = 10; // number
num = "Hello"; // string
```

2. Object-Oriented Programming

- JavaScript supports object-oriented programming concepts like:
    - **Inheritance**: Objects can inherit properties and methods from other objects.
    - **Polymorphism**: Different objects can define methods that behave differently but share the same interface.
    - **Encapsulation**: Data and methods are bundled into objects.

```
class Person {
  constructor(name) {
    this.name = name;
  }
  greet() {
    console.log(`Hello, ${this.name}`);
  }
}
let john = new Person("John");
john.greet();
```

3. Functional Programming

- JavaScript enables functional programming through:
    - **Callback Functions**: Functions passed as arguments to other functions.
    - **Higher-Order Functions**: Functions that return other functions or take functions as arguments.
    - **Closures**: Functions that can access variables from their parent scope.

```
function higherOrderFunction(callback) {
  return callback();
}
function greet() {
  return "Hello, World!";
}
console.log(higherOrderFunction(greet));
```

4. First-Class Functions

- Functions in JavaScript are treated as first-class citizens. This means:
    - Functions can be passed as arguments to other functions.
    - Functions can return other functions.
    - Functions can be stored in variables and data structures.

```
const sayHello = function() {
  return "Hello";
};
console.log(sayHello());
```

5. Prototype-Based Inheritance

- JavaScript uses prototype-based inheritance where objects inherit properties and methods from their prototypes.

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.speak = function() {
  console.log(`${this.name} makes a sound.`);
};
let dog = new Animal("Dog");
dog.speak();
```

Web Development Features

1. DOM Manipulation

- JavaScript allows developers to manipulate the Document Object Model (DOM) to dynamically update content and styles on a webpage.

```
document.getElementById("demo").innerHTML = "Hello, World!";
```

2. Event Handling

- JavaScript can listen for and handle user events like clicks, keyboard input, and mouse movement.

```
document.getElementById("btn").addEventListener("click", () => {
  alert("Button clicked!");
});
```

3. Fetch API / HTTP Requests

- JavaScript enables asynchronous data fetching using the Fetch API to interact with servers.

```
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data));
```

4. Web Storage

- JavaScript provides options for client-side storage:
  - **LocalStorage**: Stores data with no expiration.
  - **SessionStorage**: Stores data for the duration of a session.
  - **Cookies**: Stores small pieces of data with optional expiration.

```
localStorage.setItem("key", "value");
console.log(localStorage.getItem("key"));
```

Modern Features of JavaScript

1. Async/Await

- Allows writing asynchronous code in a synchronous style using `async` and `await`.

2. Promises

- Promises handle asynchronous operations, replacing callback hell with a cleaner syntax.

3. Classes

- Introduced in ES6, classes simplify object-oriented programming.

4. Modules

- JavaScript supports modular code with `import` and `export`.

5. TypeScript

- TypeScript is a superset of JavaScript that introduces static typing for better code quality.

6. JSON (JavaScript Object Notation)

- JSON is used to interchange data between systems.

7. Error Handling

- JavaScript provides `try`, `catch`, and `finally` for handling runtime errors.

8. Regular Expressions

- Regular expressions are patterns used for string matching and searching.

Not Defined vs Undefined

- **Not Defined**: A variable that has not been declared anywhere in the code.
- **Undefined**: A declared variable that has not been assigned any value.

```
console.log(a); // ReferenceError: a is not defined
let b;
console.log(b); // undefined
```

Rules for Naming Variables in JavaScript

1. **Variable names must begin with a letter, underscore `_`, or a dollar sign `$`.**
   - □ Example:
   - `let name = "John";`
   - `let _age = 25;`
   - `let $price = 100;`
2. **Variable names cannot start with numbers.**
   - □ Example:
   - `let 1name = "John"; // Error`
3. **Variable names can contain letters, digits, underscores, and dollar signs (alphanumeric characters).**
   - □ Example:
   - `let user1 = "Alice";`
   - `let _user = "Bob";`
   - `let $user = "Charlie";`
4. **JavaScript variable names are case-sensitive.**
   - `myVar` and `myvar` are two different variables.
   - □ Example:
   - `let myVar = "Hello";`
   - `let myvar = "Hi";`
   - `console.log(myVar); // Outputs: Hello`
   - `console.log(myvar); // Outputs: Hi`
5. **Avoid using reserved keywords as variable names.**
   - Reserved words include: `var`, `let`, `function`, `if`, `else`, etc.
   - □ Example:
   - `let let = "John"; // Error`
   - `let function = "Alice"; // Error`
6. **Variable names should be meaningful and descriptive.**
   - Always use names that indicate the purpose of the variable.
   - □ Good Example:
   - `let firstName = "John";`
   - `let totalAmount = 500;`
   - □ Bad Example:
   - `let x = "John";`
   - `let y = 500;`

Best Practices for Declaring Variables

1. Use `let` and `const` instead of `var` (modern best practice).
   - `let` allows re-assignment, while `const` is for constants (values that don't change).
   - □ Example:
   - `let age = 25;`
   - `age = 26; // Allowed`
   -
   - `const PI = 3.14;`
   - `PI = 3.15; // Error: Can't reassign a constant`
2. Use camelCase for variable names.
   - □ Example: `myVariableName`
   - □ Example: `my_variable_name` (not common in JavaScript).

3. Use `const` by default unless you know the value will change.
    - This helps prevent accidental reassignments.

Illegal Variable Examples

```
let 123name = "Error";  // Cannot start with a number
let my-variable = "Error";  // Hyphens are not allowed
let var = "Error";  // Reserved keyword
let @user = "Error";  // Special characters not allowed
```

Rules for Variable Declaration

1. **var**
    - Function-scoped.
    - Can be re-declared and updated.
    - Variables declared with `var` are hoisted but initialized to `undefined`.

```
console.log(a); // undefined
var a = 10;
```

2. **let**
    - Block-scoped.
    - Cannot be re-declared but can be updated.
    - Variables declared with `let` are hoisted but not initialized, resulting in a **Temporal Dead Zone (TDZ)**.

```
console.log(a); // ReferenceError
let a = 10;
```

3. **const**
    - Block-scoped.
    - Cannot be updated or re-declared.
    - Must be initialized during declaration.
    - Variables declared with `const` are also subject to the Temporal Dead Zone (TDZ).

```
const z = 10;
// z = 20; // Error: Assignment to constant variable.
```

4. **General Rules**:
    - Always prefer `let` and `const` over `var` for modern JavaScript.
    - Use `const` when the value should remain unchanged.
    - Use `let` when reassignment is needed.
    - Avoid using `var` due to its function-scoping and hoisting behavior.