**Java-script Theory**

**1.What is JavaScript? Explain the role of JavaScript in web development.**

**Ans.** JavaScript is a high-level, interpreted programming language primarily used to create interactive effects and enhance the functionality of web pages. It is one of the core technologies of the World Wide Web, alongside HTML (HyperText Markup Language) and CSS (Cascading Style Sheets).Originally developed by Netscape in the mid-1990s, JavaScript has evolved significantly and is now standardized by the ECMAScript specification.

**2. How is JavaScript different from other programming languages like Python or Java?**

**Ans.**

**3. Discuss the use of <script > tag in HTML. How can you link an external JavaScript file to an HTML document?**

**Ans.** The <script> tag in HTML is used to add JavaScript code to a web page. JavaScript is a scripting language that adds interactivity, dynamic behavior, and functionality to static HTML content. The <script> tag can either include JavaScript code directly inside the HTML file or link to an external JavaScript file.

This tag plays a crucial role in web development as it enables client-side programming, allowing developers to create interactive features like sliders, form validation, content updates without page reloads (AJAX), and more.

**4. What are variables in JavaScript? How do you declare a variable using var, let, and const?**

**Ans.** Variables in JavaScript are containers for storing data values. They allow programmers to store, retrieve, and manipulate data within a program. A variable can hold various types of data such as numbers, strings, objects, arrays, and more. JavaScript is a dynamically typed language, so variables do not need a fixed data type and can hold different types of values at different times.

**5. Explain the different data types in JavaScript. Provide examples for each.**

**Ans. 1. Primitive Data Types**

**Primitive data types are basic, immutable values. They are stored directly in the variable and not as references.**

* **String:** Represents a sequence of characters used for textual data. Strings are written inside quotes, either single or double.
* **Number:** Represents both integers and floating-point numbers. JavaScript does not differentiate between integers and decimals.
* **Boolean:** Represents logical values – either true or false.
* **Undefined:** A variable that has been declared but not assigned a value is of type undefined.
* **Null:** A special data type that represents the intentional absence of any object value. It means "no value".
* **Symbol:** Introduced in ES6, it is used to create unique identifiers, often for object properties.
* **BigInt:** Introduced to represent integers that are too large for the Number type.

**6. What is the difference between undefined and null in JavaScript?**

**Ans.** **Type:** undefined (primitive)

**Definition:** Represents the absence of a value — typically when a variable is declared but not assigned a value.

Automatically assigned by JavaScript.

**Common Cases:**

* Uninitialized variables
* Missing function parameters
* Missing object properties
* Return value of functions with no return

**null**

* **Type**: object (this is a known historical bug in JavaScript)
* **Definition**: Represents the **intentional absence of any object value**.
* **Explicitly assigned** by the programmer.
* **Common Cases**:
  + Resetting a variable
  + Representing an empty object reference
  + Placeholder for a value that will be later assigned

**7. What are the different types of operators in JavaScript? Explain with examples.**

**Ans. a) Arithmetic operators :**

**Definition:** Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, etc.

**Operators & Purpose:**

* **+ :** Adds two numbers.
* **- :** Subtracts the second number from the first.
* **\* :** Multiplies two numbers.
* **/ :** Divides the first number by the second.
* **% :** Returns the remainder after division.
* **\*\* :** Performs exponentiation (power).
* **++ :** Increments a number by 1.
* **-- :** Decrements a number by 1.

**b) Assignment operators :**

**Definition:** Assignment operators are used to assign values to variables. They can also perform operations before assignment.

**Operators & Purpose:**

* **= :** Assigns a value to a variable.
* **+= :** Adds a value and assigns the result.
* **-= :** Subtracts a value and assigns the result.
* **\*= :** Multiplies and assigns the result.
* **/= :** Divides and assigns the result.
* **%= :** Takes modulus and assigns the result.

**c) Comparison operators :**

**Definition:** Comparison operators are used to compare two values and return a Boolean result (true or false).

**Operators & Purpose:**

* **== :** Equal to (loose comparison, type conversion allowed).
* **=== :** Strict equal to (no type conversion).
* **> :** Greater than.
* **< :** Less than.
* **>= :** Greater than or equal to.
* **<= :** Less than or equal to.

**d) Logical operators :**

**Definition:** Logical operators are used to combine or invert Boolean expressions.

**Operators & Purpose:**

* **&& (AND) :** Returns true only if both conditions are true.
* **|| (OR) :** Returns true if at least one condition is true.
* **! (NOT) :** Inverts the Boolean value (true becomes false, and vice versa).

**8.What is the difference between == and === in JavaScript?**

**Ans. == (Loose Equality)**

* Compares values after converting them to a common type.
* Performs type coercion if the types are different.
* Can lead to unexpected results due to automatic conversion.

**=== (Strict Equality)**

* Compares both value and type.
* Does not perform type coercion.
* Preferred in most cases for predictable comparisons.

**9.What is control flow in JavaScript? Explain how if-else statements work with an example.**

**Ans.** Control flow in JavaScript refers to the order in which statements and instructions are executed in a program. By default, JavaScript runs code from top to bottom, but control flow statements allow you to change that order — for example, by making decisions (if statements), looping through code (for, while), or jumping out of code blocks (break, return).

**if-else Statements in JavaScript**

The if-else statement is used to make decisions in your code based on conditions. It allows your program to execute certain blocks of code only if a specific condition is true, and optionally do something else if the condition is false.

**10.Describe how switch statements work in JavaScript. When should you use a switch statement instead of if-else?**

**Ans.** A switch statement is used to execute different blocks of code based on the value of a variable or expression. It is an alternative to writing multiple if-else statements when you're comparing the same variable to many different values.

|  |  |
| --- | --- |
| **Use switch when...** | **Use if** **when...** |
| You're comparing the same variable or expression to many values | You have complex conditions or need to check different variables |
| You want cleaner, more readable code for multiple cases | Conditions involve logical operators (&&, ` |
| All comparisons are equality checks | Conditions are based on ranges, functions, or calculations |

**11.Explain the different types of loops in JavaScript (for, while, do-while). Provide a basic example of each.**

**Ans. 1] for Loop**

**Concept/Theory:**

* Best used when you know ahead of time how many times you want to loop.
* It includes three parts in one line:  
  initialization; condition; increment/decrement.

**2] while Loop**

**Concept/Theory:**

* Use when you **don't know how many times** to loop.
* The loop continues **as long as the condition is true**.
* The condition is **checked before** each loop iteration.

**3] do-while Loop**

**Concept/Theory:**

* Similar to while, **but always runs the code at least once**.
* The condition is **checked after** the code executes.

**12.What is the difference between a while loop and a do-while loop?**

**Ans.**

|  |  |  |
| --- | --- | --- |
| **Feature** | **while Loop** | **do-while Loop** |
| **Condition Check** | Checked before the loop runs | Checked after the loop runs |
| **Minimum Execution** | May not run at all if condition is false | Runs at least once |
| **Use Case** | Use when you only want to run the loop if the condition is already true | Use when you must run the code at least once regardless of condition |

**13.What are functions in JavaScript? Explain the syntax for declaring and calling a function.**

**Ans.** A function in JavaScript is a reusable block of code that performs a specific task. Functions help make your code modular, organized, and easier to maintain.

**14.What is the difference between a function declaration and a function expression?**

**Ans. 1. Function Declaration**

**Definition (Theory):**A function declaration is a way to define a named function using the function keyword at the top level of code structure.

**Key Theoretical Characteristics:**

* **Hoisted:** Entire function is moved to the top of its scope at runtime, so it can be called before it's defined.
* Has a fixed name.
* Used for defining functions that are intended to be reused throughout the code.

**2. Function Expression**

**Definition (Theory):**A function expression defines a function and assigns it to a variable, possibly anonymously**.**

**Key Theoretical Characteristics:**

* **Not hoisted:** Only the variable is hoisted, not the function body, so you cannot call it before it’s defined.
* Can be anonymous or named, but the name (if given) is usually local to the function.
* Treated like a value — can be assigned to variables, passed to other functions, or returned.

**15. Discuss the concept of parameters and return values in functions.**

**Ans. 1. Parameters**

**What Are Parameters?**

* Placeholders or variables listed in the function definition.
* They define what inputs the function expects.
* You can pass arguments (actual values) when calling the function.

**2. Return Values**

**What Is a Return Value?**

* A value sent back by the function to the place where it was called.
* Done using the return keyword.
* Allows the function to produce a result instead of just performing an action

**16. What is an array in JavaScript? How do you declare and initialize an array?**

**Ans.** An array in JavaScript is a data structure that allows you to store multiple values in a single variable. These values can be of any type: numbers, strings, objects, other arrays, etc.

**Key Features of Arrays:**

* **Ordered:** Elements are stored in a sequence and can be accessed using an index.
* **Indexed:** The first element has an index of 0, the second is 1, and so on.
* **Dynamic:** Arrays can grow or shrink in size.

**17. Explain the methods push(), pop(), shift(), and unshift() used in arrays.**

**Ans. 1) push :** Adds one or more elements to the end of an array.

**2) pop :** Removes the last element from an array and returns it.

**3) shift :** Removes the first element from an array and returns it.

**4) unshift :** Adds one or more elements to the beginning of an array.

**18. What is an object in JavaScript? How are objects different from arrays?**

**Ans.** An object in JavaScript is a collection of key-value pairs used to store and organize data. It allows you to represent real-world entities with properties and behaviors.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Object** | **Array** |
| **Purpose** | Store **named data** as key-value pairs | Store **ordered lists** of items |
| **Access** | By **property name** | By **index number** |
| **Structure** | { key: value } | [value1, value2, value3] |
| **Key Type** | Keys are **strings** (or symbols) | Keys are **numeric indices** |
| **Best use case** | When data needs to be labeled (e.g., user profile) | When working with a list or sequence of values |

**19. Explain how to access and update object properties using dot notation and bracket notation.**

**Ans. Definition:**Bracket notation allows you to access and update properties using a string value, even if it's stored in a variable. It’s more flexible than dot notation.

**Characteristics:**

* Allows dynamic property names
* Supports property names with spaces or special characters
* Required when using variables to refer to property names

**20. What are JavaScript events? Explain the role of event listeners.**

**Ans.** JavaScript events are actions or occurrences that happen in the browser, often triggered by user interactions or browser behavior. JavaScript can "listen" for these events and respond with specific code (functions).

**Common Examples of Events:**

* **click –** when a user clicks an element
* **submit –** when a form is submitted
* **keydown –** when a key is pressed
* **mouseover –** when the mouse hovers over an element
* **load –** when a web page finishes loading

**21. How does the addEventListener() method work in JavaScript? Provide an example.**

**Ans.** The addEventListener() method is used to attach an event handler to a specific HTML element. It allows you to listen for specific events (like clicks, key presses, etc.) and define a callback function that runs when that event occurs.

**22. What is the DOM (Document Object Model) in JavaScript? How does JavaScript interact with the DOM?**

**Ans.** The DOM (Document Object Model) is a programming interface for web documents. It represents the structure of an HTML or XML document as a tree of objects, where each HTML element becomes a node (object) in that tree.Think of the DOM as a live map of your web page, which JavaScript can access and manipulate in real time**.**

JavaScript interacts with the DOM (Document Object Model) to make web pages dynamic and interactive. It does this by using the DOM API, which provides methods and properties to access, manipulate, and respond to changes in the HTML document.

**23. Explain the methods getElementById(), getElementsByClassName(), and querySelector() used to select**

**elements from the DOM.**

**Ans. 1. getElementById()**

**Description:**

* Selects a single element by its ID attribute.
* Returns the first matching element (IDs should be unique).

**2. getElementsByClassName()**

**Description:**

* Selects all elements with a specific class name.
* Returns an HTMLCollection (like an array, but not exactly).

**3. querySelector()**

**Description:**

* Selects the **first element** that matches a **CSS selector**.
* Very flexible: can use ID (#id), class (.class), tag, or complex CSS selectors.

**24. Explain the setTimeout() and setInterval() functions in JavaScript. How are they used for timing events?**

**Ans. setTimeout()**

* The setTimeout() method is used to execute a function once after a specified number of milliseconds.
* It does not block other code from running while waiting.
* Returns a unique timeout ID which can be used to cancel it using clearTimeout().

**setInterval()**

* The setInterval() method is used to repeatedly execute a function at specified intervals (in milliseconds).
* It continues to run until it is explicitly stopped using clearInterval() with its ID.
* Useful for creating recurring tasks like clocks, sliders, or polling servers.

**25. Provide an example of how to use setTimeout() to delay an action by 2 seconds.**

**Ans.** The setTimeout() function in JavaScript is used to delay the execution of a specific function or block of code by a given number of milliseconds.

setTimeout(function, delayInMilliseconds);

**26. What is error handling in JavaScript? Explain the try, catch, and finally blocks with an example.**

**Ans.** Error handling in JavaScript is the process of catching and managing errors that occur while the script is running. Without error handling, JavaScript will stop executing if an error occurs, which can cause a poor user experience.

**To manage errors gracefully, JavaScript provides a structured way using:**

* try block
* catch block
* finally block (optional)

**1. try Block**

* The try block wraps code that might throw an error.
* If an error occurs inside try, JavaScript jumps to the catch block.

**2. catch Block**

* The catch block handles the error.
* It has access to the error object, which gives details about what went wrong.

**27. Why is error handling important in JavaScript applications?**

**Ans. 1. Prevents Application Crashes**

Without proper error handling, runtime errors can cause JavaScript to stop executing, leading to broken functionality or a completely frozen webpage. Handling errors allows the app to recover gracefully and continue working.

**2. Improves User Experience**

When errors are caught and managed, users can receive friendly messages or fallback options instead of seeing cryptic errors or a blank screen. This helps keep users informed and reduces frustration**.**

**3. Helps Debugging and Maintenance**

Error handling provides detailed information about what went wrong, making it easier for developers to identify, log, and fix issues quickly.

**4. Ensures Security**

Uncaught errors might expose sensitive internal details or cause unexpected behavior that attackers can exploit. Proper handling helps contain errors and protect the application.

**5. Supports Complex Logic**

Many applications rely on asynchronous operations (like network requests), user inputs, or external APIs that may fail unpredictably. Error handling allows these cases to be managed smoothly without breaking the app.