Name : Patel Om Jigneshkumar

Course : Node JS

Faculty : Rinkal Ma’am

Assignment : Introduction to Javascript

Q1 : What is the difference between Java & JavaScript?

→Java

- **Java** is an object-oriented programming language and has a virtual machine platform that allows you to create compiled programs that run on nearly every platform. Java promised, “Write Once, Run Anywhere”.

JavaScript

- JavaScript was created in the first place for DOM manipulation. Earlier websites were mostly static, after JS was created dynamic Web sites were made.
- Functions in JS are objects. They may have properties and methods just like other objects. They can be passed as arguments in other functions.
- Can handle date and time.
- Performs Form Validation although the forms are created using HTML.
- No compiler is needed.

Q2. What is JavaScript?

→JavaScript is a single-threaded, synchronous programming and scripting language widely used in web development. It plays a crucial role in both front-end and back-end development, enabling dynamic and interactive user experiences. Unlike compiled languages, JavaScript is interpreted through web browsers, making it highly accessible and versatile. It is a loosely typed (or weakly typed) language, meaning variable types are determined at runtime.

Q3. What are the data types supported by JavaScript?

→In JavaScript, there are several data types that are used to represent different kinds of values. Here are the main data types along with examples:

1. **Number**: Represents numeric values, including integers and floating-point numbers.

```
let age = 25;
let temperature = 98.6;
```

2. **String**: Represents textual data enclosed within single or double quotes.

   ```
   let name = "John Doe";
   let message = 'Hello, world!';
   ```

3. **Boolean**: Represents a logical value indicating true or false.

   ```
   let isStudent = true;
   let isEmployed = false;
   ```

4. **Undefined**: Represents a variable that has been declared but has not been assigned a value.

   ```
   let x;
   console.log(x); // Outputs: undefined
   ```

5. **Null**: Represents the intentional absence of any object value.

   ```
   let car = null;
   ```

6. **Object**: Represents a collection of key-value pairs.

   ```
   let person = {
   name: "Alice",
   age: 30,
   isStudent: false
   };
   ```

7. **Array**: Represents a collection of elements, indexed by integers.

```
let numbers = [1, 2, 3, 4, 5];
let fruits = ['apple', 'banana', 'orange']
```

Q4. What are the scopes of a variable in JavaScript?

→

- **Block Scope**

ES6 introduced two important new JavaScript keywords: let and const.

These two keywords provide **Block Scope** in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block:

Example

```
{
    var v1 = "hello";
    const v2 = "world";
    let x=2;
    x*=2;
    console.log(x);
    console.log(variable_2);
}

console.log(variable_1);
```

- **Local Scope**

Variables declared within a JavaScript function, are **LOCAL** to the function:

Since local variables are only recognized inside their functions, variables with the same name can be used in different functions.

Local variables are created when a function starts, and deleted when the function is completed.

Example:

```
// This part of code cannot use firstName
function myFunction() {
 let firstName = "Krishna";
 // This part of code can use firstName
}
This part of code cannot use firstName
```

- **Function Scope**

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with var, let and const are quite similar when declared inside a function.

They all have Function Scope.

Example:

```
function myFunction() {
  let firstName = "Om";   // Function Scope
}
```

Q5. What is Callback?

→

- **Callback :**

A callback is a function that is passed as an argument to another function and is executed after the completion of that main function. In simple terms, a callback function is called at the end of a task to either deliver results or perform an action.

- **Use of Callback:**

Callbacks are used for managing the outcomes of asynchronous tasks without blocking the program's execution. Asynchronous tasks, like network requests or database queries, take time to finish. If these tasks were synchronous, the program would halt until they were done, resulting in a sluggish user experience.

Example:

```
function mainFunction(callback) {
  console.log("Performing operation...");
  setTimeout(function() {
    callback("Operation complete");
  }, 1000);
}
function callbackFunction(result) {
  console.log("Result: " + result);
}
mainFunction(callbackFunction);
```

Q6. What is Closure? Give an example.

→Closures in JavaScript are functions that retain access to variables from their containing scope even after the parent function has finished executing.

```
function fn() {
    let b = 1;
    function inner() {
        return b;
    }
    return inner;
}
let get_func_inner = fn();

console.log(get_func_inner());
console.log(get_func_inner());
console.log(get_func_inner());
```

Output

```
1
1
1
```

Q7. What is the difference between the operators '==' & '===''?

→

- **JavaScript '==' operator:**

  In Javascript, the '= =' operator is also known as the loose equality operator which is mainly used to compare two values on both sides and then return true or false. This operator checks equality only after converting both the values to a common type i.e type coercion.

  Example:

```
<script>
        console.log(21 == 21);
        console.log(21 == '21');
        console.log('food is love'=='food is love');
        console.log(true == 1);
        console.log(false == 0);
        console.log(null == undefined);
</script>
```

  All of them will return true as output.

- **JavaScript '===' operator:**
  Also known as strict equality operator, it compares both the value and the type which is why the name "strict equality".

  Example:

```
<script>
      // '===' operator
      console.log('hello world' === 'hello world');
      console.log(true === true);
      console.log(5 === 5);
</script>
```

Q8. What is the difference between null & undefined?

→

- **Undefined**
  undefined is a primitive value automatically assigned to variables in certain situations:
  - **Uninitialized Variables:** A variable that is declared but not initialized will have the value undefined.
  - **Missing Function Return:** If a function does not explicitly return a value, it returns undefined by default.
  - **Non-Existent Object Properties or Array Elements:** Accessing an object property or an array element that does not exist results in undefined.

  ### Example:

```
let x; // variable declared but not initialized
console.log(x); // Output: undefined

function hello() {
// no return statement, so the function returns undefined
}
console.log(hello()); // Output: undefined

let obj = {};
console.log(obj.property); // Output: undefined
```

- **Null**
  null is a special value in JavaScript that represents the deliberate absence of any object value. It is often used to:

  - **Indicate "No Value":** null explicitly shows that a variable or object property should have no value.
  - **Reset or Clear Variables:** Use null to reset a variable or remove its reference to an object.

  ### Example:

```
let y = null; // variable set to null
console.log(y); // Output: null

let obj = { property: null }; // set to null
console.log(obj.property); // Output: null
```

Q9. What would be the result of 2+5+”3″?

→The result of the expression 2 + 5 + "3" in JavaScript would be "73".

- JavaScript evaluates the expression from left to right.
- First, it adds 2 + 5, which results in 7 (since both are numbers).
- Then, it encounters the string "3". In JavaScript, when you add a number to a string, it performs type coercion, converting the number to a string.

- So, 7 + "3" becomes "7" + "3", which results in the string "73".
- Thus, the result is the string "73".

Q10. What is the difference between Call & Apply?

→The key difference between call() and apply() in JavaScript is how they handle arguments when invoking a function.

call():

- **Syntax**: func.call(thisArg, arg1, arg2, ...)
- The call() method invokes a function with a specified this value and **individual arguments**.
- You pass arguments **separately** to the function.

Example:

```
function greet(greeting, punctuation) {
  console.log(`${greeting}, ${this.name}${punctuation}`);
}
const person = { name: 'Patel' };// Using call(), pass arguments separately

greet.call(person, 'Hello', '!');
```

apply():

- **Syntax**: func.apply(thisArg, [argsArray])
- The apply() method invokes a function with a specified this value and **arguments as an array**.
- You pass arguments as **an array or array-like object**.

Example:

```
function greet(greeting, punctuation) {
  console.log(`${greeting}, ${this.name}${punctuation}`);
}

const person = { name: 'Patel' };

// Using apply(), pass arguments as an array
greet.apply(person, ['Hello', '!']);
```