Web Technology

UNIT – III:

Scripting:

JavaScript: Introduction, documents, forms, statements, functions, objects, Introduction to AJAX.

Networking:

Internet Addressing, InetAddress, Factory Methods, Instance Methods, TCP/IP Client Sockets, URL, URL Connection, TCP/IP Server Sockets, Datagram.

AKTU PYQs

Web Technology (AKTU 2022 – 23)

1. Discuss about Math and Date objects in JavaScript. Write a program in JavaScript to display digital clock showing HH:MM:SS.

Web Technology (AKTU 2021 – 22)

1. Create Object in JavaScript.
2. Explain the working of AJAX along with its applications. Mention a suitable example.
3. Design a JavaScript program to display the current day and time in the following format.
   Today is : Monday.
   Current time is : 11 AM : 50 : 58

Web Technology (AKTU 2020 – 21)

1. Explain JavaScript. Describe various objects in JavaScript.
2. Explain Function in JavaScript with suitable example.

Web Technology (AKTU 2019 – 20)

1. Discuss AJAX. Explain the application of AJAX with the help of suitable examples.
2. Compare Java and JavaScript. Explain and demonstrate 5 different types of objects in JavaScript with example.

Web Technology (AKTU 2018 – 19)

1. Compare Java and JavaScript. Write a JavaScript program to define a user defined function for sorting the values in an array.
2. Create an html page named as “String_Math.html” and within the script tag define some string variables and use different string function to demonstrate the use of the predefined functions. Do the same for the Math function.

AKTU PYQs

Web Designing (AKTU 2023 – 24)

1. Discuss the data types in JavaScript.
2. Explain the four types of loops in JavaScript.
3. Define JavaScript operators. What the different types of operators in JavaScript? Explain.
4. What are the three kinds of popup boxes in JavaScript? Explain.
5. What is array? Define its properties and methods.

Web Designing (AKTU 2022 – 23)

1. List out the various dialog boxes/popup boxes in Java Script..
2. Describe the use of typeof operator in JavaScript.
3. Design a HTML form with name, address, pincode and password. Outline a JavaScript code that can validate on following constraints:
   a. No fields must be left blank
   b. Pincode must be numeric and contain 6 digits
   c. Password field
   (i) At least contains 1 Capital letter.
   (ii) Minimum length 8 characters.
4. Explain the various event handlers in JavaScript with an example of each.
5. Create a JavaScript program to make arithmetic calculator.
6. Discuss function in Java Script. Outline a JS function that takes 4 variables, a, b, c, d, and prints true in an alert box if input a > input b and input c > input d. Else, print false in an alert box.

AKTU PYQs

Web Designing (AKTU 2021 – 22)

1. List the various dialog boxes in JavaScript.
2. Write a JavaScript function to find the sum of first 20 even natural numbers.
3. Write a JavaScript that takes three integers from the user and displays the sum, average, product, smallest and largest of the numbers in an alert dialog.
4. What are the methods associated with the array object in JavaScript? Explain each one with an example.
5. Explain the various event handlers in JavaScript. Give an example of each. Write a JavaScript program to develop the arithmetic calculator.
6. Write a JavaScript program to perform the validation process in an application program.

Web Designing (AKTU 2020 – 21)

1. Differentiate between Java and JavaScript.
2. What is 'this' keyword in JavaScript.
3. Write a JavaScript program to print first N odd numbers divisible by 7.
4. Write the code for adding new elements dynamically using JavaScript.
5. What are the various JavaScript Objects? Explain each with an example.
6. Discuss how do you use JavaScript for form validation? Develop a complete applicationthat would include information functions to validate the user data.
7. Write short notes on the operators in JavaScript.

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-1

Today's Target

- JavaScript
  - Introduction
    - Features
    - Java and JavaScript: A comparison
    - The first program – "Hello World"
  - AKTU PYQs

By Amol Sharma sir

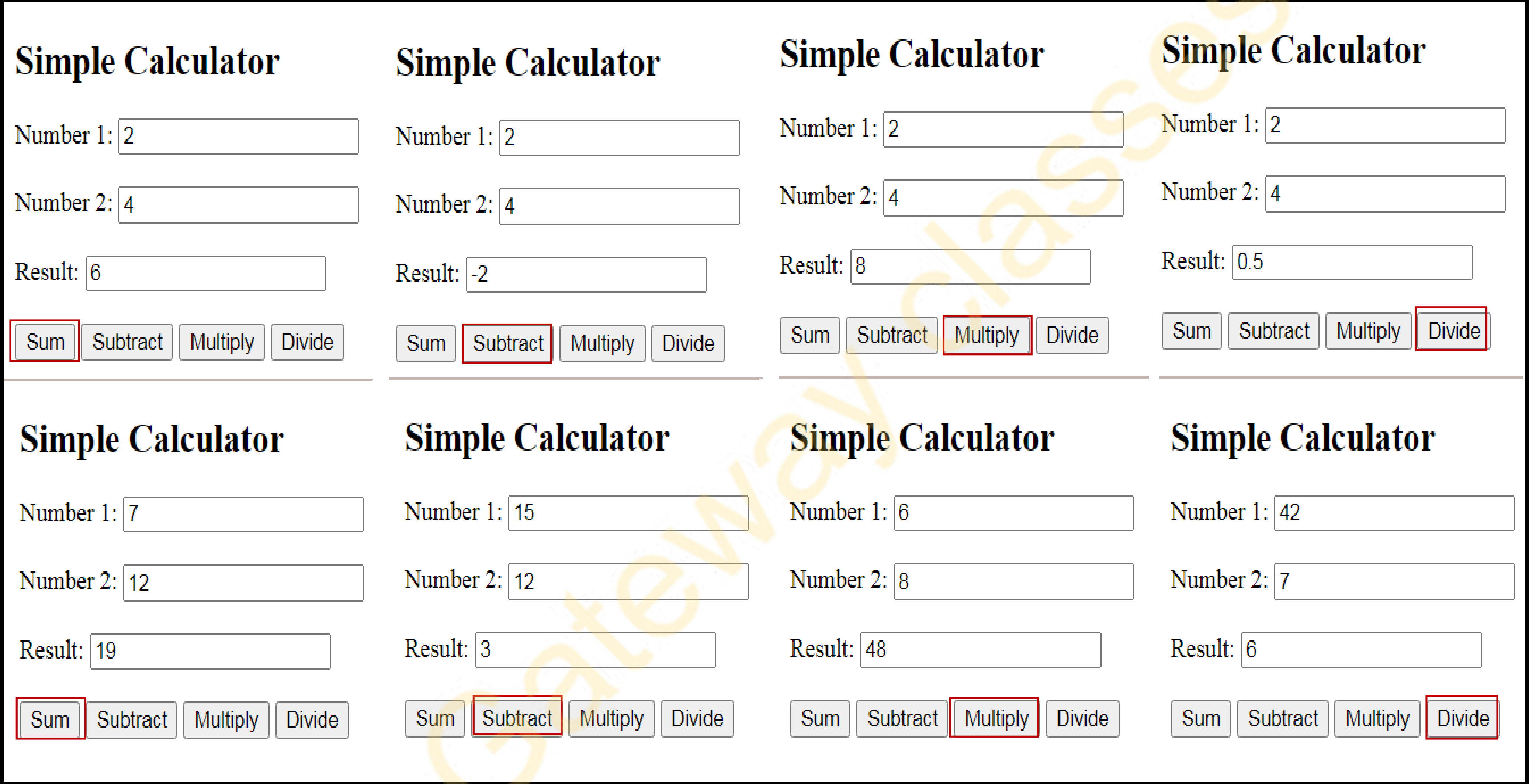
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

"JavaScript is a lightweight interpreted programming language used to create dynamic and interactive elements in web applications."

- **Lightweight**
  JavaScript is lightweight, means it executes tasks directly in the browser, allowing web pages to be dynamic and responsive without heavy processing i.e. without demanding much processing power or memory.
- **Interpreted**
  JavaScript is an interpreted language, meaning code runs directly in the browser without prior compilation, enabling quicker testing and execution.
- **Programming Language**
  As a programming language, JavaScript can create complex logic, control behavior, and interact with HTML/CSS to build interactive web applications.
- **Dynamic and Interactive Elements**
  JavaScript enables dynamic and interactive elements like real-time updates, animations, and user-driven actions, enhancing user engagement on web pages.

JavaScript



Simple Calculator

Number 1: 2

Number 2: 4

Result: 6

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 2

Number 2: 4

Result: -2

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 2

Number 2: 4

Result: 8

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 2

Number 2: 4

Result: 0.5

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 7

Number 2: 12

Result: 19

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 15

Number 2: 12

Result: 3

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 6

Number 2: 8

Result: 48

Sum | Subtract | Multiply | Divide

Simple Calculator

Number 1: 42

Number 2: 7

Result: 6

Sum | Subtract | Multiply | Divide

Interactive

Webpage responds based on user actions.

Dynamic

Output changes when the inputs change.

Programming

Computing Sum, Subtraction, Multiplication, Division requires program (script) to be written to implement logic.

JavaScript

Points to note:

- JavaScript makes web pages alive.
- The programs in this language are called scripts. They can be written right in a web page's HTML and run automatically as the page loads.
- JavaScript was created by Brendan Eich at Netscape in 1995. It initially had the name: "LiveScript" but Java was very popular at that time, so the name was changed to JavaScript. But JavaScript is a fully independent language with its own specification and *has no relation to Java at all*.

JavaScript

Points to note:

Java and JavaScript: A comparison

- Java is a general-purpose, compiled language used for building a wide range of applications, including: Desktop applications, Web servers, Scientific computing and Android apps.
- JavaScript is a scripting language primarily used for creating interactive web pages and web applications. It can also be used for server-side development with Node.js.

- Java is influenced by C++, with a more verbose syntax and strong emphasis on object-oriented programming (OOP) principles.
- JavaScript is influenced by C with a more flexible syntax and support for various programming paradigms, including OOP, functional, and imperative.

- Java is compiled into bytecode, which is then executed by the Java Virtual Machine (JVM).
- JavaScript is interpreted directly by the browser or other runtime environments.

JavaScript

Points to note:

Java and JavaScript: A comparison

- Java is purely object-oriented language, where everything is an object (except for primitive data types).
- JavaScript supports OOP but also allows for other programming styles.

- Java runs on any platform with a JVM installed.
- JavaScript runs primarily in web browsers, but can also be used in other environments like Node.js.

- Java is statically typed, meaning the data type of a variable must be declared at compile time.
- JavaScript is dynamically typed, meaning the data type of a variable is determined at runtime.

JavaScript

Points to note:

- JavaScript continues to evolve with each new version of ECMAScript (ES), which defines the language's capabilities and features. The latest and current 15th version is ES2024.

- JavaScript can execute at client side (in the browser), as well as on the server side, or actually on any device that has a special program called the JavaScript engine.

- The browsers have JavaScript engine embedded in them. Most popular browsers Chrome, Edge, Firefox, Opera, Safari support JavaScript.

- The JavaScript engine parses scripts, converts them into machine code, and executes them quickly.

JavaScript

Writing Scripts in HTML

Ways to Embed JavaScript

- Internal: Writing scripts directly inside HTML <script> tags in the document.

  Example: `<script> document.write("Hello World!"); </script>`

- External: Linking a .js file to the HTML document for modular code.

  Example: `<script src="script.js"></script>`

Placement in HTML

- Head: Loads before content but may delay page rendering.
- Body (at the end): Commonly used for faster page loading and to ensure HTML is fully loaded before JavaScript runs.

JavaScript

The first program – *“Hello World”*

- The document.write() method writes the text directly to the HTML document or web page.

  Example: <script> document.write("Hello World”); </script>

- We can use the window alert() method to print "Hello Word" in a dialogue box.

  Example: <script> alert("Hello World“) </script>

- The console.log() prints the message to web Console.

  Example: <script> Console.log("Hello World“) </script>

- The innerHTML property of an HTML element may be used to set the text as HTML content of the element.

  Example: <script> document.getElementById("output").innerHTML = “Hello World”; </script>

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-2

Today's Target

- JavaScript
  - Variables
  - Constants
  - Datatypes
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Variables

Variables are containers for storing data values.

They act as "labels" that hold data which can be used and manipulated in a program.

Before using a variable, you need to declare it.

Variable Declaration

We can declare variables to store data by using the var, or let keywords.

Example:

```
let name;               var name;          //Old way of declaring variables
name =‘Gateway’;        name = ‘Gateway’;
```

A variable name can be any valid identifier. By default, the variable has a special value *undefined* if it is not assigned a value.

Earlier, it was technically possible to create a variable by a mere assignment of the value without using let/ var. This still works now if we don’t put ‘use strict’ in our scripts to maintain compatibility with old scripts.

JavaScript

Rules for Variable Naming

- Variable names are case-sensitive. This means that the ‘name’ and ‘Name’ are different variables.
- Variable names can only contain letters, numbers, underscores, or dollar signs and cannot contain spaces. Also, variable names must begin with a letter, an underscore (_) or a dollar sign ($).
- Variable names cannot use the reserved words.

By convention, variable names use camelcase like name, and firstName.

JavaScript is a dynamically typed language. This means that you don’t need to specify the variable’s type in the declaration like other static-typed languages such as Java or C#.

JavaScript

Differences between ‘var’ and ‘let’:

- var is function-scoped/ global-scoped whereas let is block-scoped. *Example:*

```
if (true) {        var x = 10;    }    console.log(x); // 10, accessible outside the if block
if (true) {        let y = 20;    }    console.log(y); // Error: y is not defined
```

- var allows re-declaration within the same scope whereas let does not allow re-declaration within the same scope. *Example:*

```
var a = 5; var a = 10; // No error, re-declaration allowed
let b = 5;  let b = 10; // Error: Identifier 'b' has already been declared
```

- Both var and let are hoisted to the top of their scope, but var initializes as undefined, while let is not initialized (remains in a "temporal dead zone" until declaration). *Example:* console.log(c); // undefined var c = 15;

```
console.log(d); // Error: Cannot access 'd' before initialization let d = 25;
```

JavaScript

Constants

Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error.

Example:

```
const NUM_SUBJECTS = 3;
```

When a programmer is sure that a variable will never change, they can declare it with const to guarantee and communicate that fact to everyone.

JavaScript

Datatypes

JavaScript has 8 basic data types:

1. *number:* For numbers of any kind (integer or floating-point). Integers are limited by $\pm(2^{53}-1)$

Besides regular numbers, there are “special numeric values” which also belong to this data type: *Infinity, -Infinity* and *NaN*. Infinity represents the mathematical Infinity ∞. NaN represents a computational error. It is a result of an incorrect or an undefined mathematical operation, for example: “four" / 2.

2. bigint: For integer values of arbitrary length. A BigInt value is created by appending n to the end of an integer:

1234567890123456789012345678901234567890n;

3. string: For text. A string may contain zero or more characters; there is no separate single-character type.

JavaScript

Datatypes

4. boolean: For logical values, true or false.

5. null: Represents an intentional absence of any value. It's a standalone type with a single value null.

6. undefined: Indicates an unassigned or unknown value. It’s a standalone type with a single value undefined.

7. object: For more complex data structures, such as arrays, functions, and objects themselves.

8. symbol: For unique identifiers, primarily used for object properties to ensure uniqueness.

JavaScript

Datatypes

Using the ‘typeof’ Operator:

The typeof operator shows the type of a given variable.

Syntax: typeof x (or typeof(x)).

It returns a string, like "string" or "number".

Note: The result of typeof null is "object". That’s an officially recognized historical error in typeof, operator of JavaScript and kept for compatibility.

null is not an object. It is a special value with a separate type of its own. The behavior of typeof is wrong here.

JavaScript

Datatypes

Example:

```
<script>
    document.writeln(typeof undefined+"<br>")                  // "undefined"
    document.writeln(typeof 0+"<br>")                          // "number"
    document.writeln(-4/0+"<br>")                              // "-Infinity"
    document.writeln('four'/2+"<br>")                          // "NaN"
    document.writeln(typeof -Infinity+"<br>")                  // "number"
    document.writeln(typeof Infinity+"<br>")                   // "number"
    document.writeln(typeof NaN+"<br>")                        // "number"
    document.writeln(typeof 0+"<br>")                          // "number"
    document.writeln(typeof 20n+"<br>")                        // "bigint"
    document.writeln(typeof true+"<br>")                       // "boolean"
    document.writeln(typeof "Gateway"+"<br>")                  // "string"
    document.writeln(typeof Symbol("symbol1")+"<br>")          // "symbol"
    document.writeln(typeof document+"<br>")                   // "object"
    document.writeln(typeof null+"<br>")                       // "object"
    document.writeln(typeof alert+"<br>")                      // "function"
</script>
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-3

Today's Target

- JavaScript Operators
  - Arithmetic,
  - Comparison,
  - Logical,
  - Assignment,
  - Bitwise,
  - Conditional,
  - Nullish Coalescing
- AKTU PYQs

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Operators

In JavaScript, an operator is a symbol used to perform operations on one or more operands, such as variables or values, and it produces a result.

For example, in the expression 4 + 5, 4 and 5 are operands, while + is the operator that adds them, resulting in 9.

Arithmetic Operators

Arithmetic operators in JavaScript perform mathematical calculations on numeric values (operands). Most arithmetic operators are binary operators, means that they perform calculations on two operands. However, some arithmetic operators are unary, meaning they operate on just a single operand.

JavaScript

Operators

Arithmetic Operators

Operator	Name	Description
+	Addition	Adds two operands. It is also used for string concatenation.
-	Subtraction	Subtracts the second operand from the first
*	Multiplication	Multiply both operands
/	Division	Divide the numerator by the denominator
%	Modulus	Outputs the remainder of an integer division
++	Increment	Increases an integer value by one
--	Decrement	Decreases an integer value by one
**	Exponentiation	Returns the power of the first operand raised to the second.

JavaScript

Operators

Comparison Operators

In JavaScript, comparison operators are used to compare two variables or values, resulting in a boolean value of either true or false based on the outcome.

For instance, comparison operators help determine if two operands are equal.

- These operators are commonly used within logical expressions, which evaluate to either true or false.
- Comparison operators are binary, meaning they operate on two operands. The operands can be numbers, strings, booleans, or objects.

JavaScript

Operators

Comparison Operators

Operator	Name	Description
= =	Equal	Checks if two values are equal, allowing type conversion.
!=	Unequal	Checks if two values are not equal, allowing type conversion.
= = =	Strictly equal	Checks if two values are equal and of the same type (strict equality).
!= =	Strictly unequal	Checks if two values are not equal or not of the same type (strict inequality).
>	Greater than	Checks if the left value is greater than the right value.
<	Less than	Checks if the left value is less than the right value.
>=	Greater than or Equal to	Checks if the left value is greater than or equal to the right value.
<=	Less than or Equal to	Checks if the left value is less than or equal to the right value.

JavaScript

Operators

Logical Operators

Logical operators in JavaScript are symbols that help you make decisions in code by checking whether conditions are true or false. They allow you to test multiple conditions at once or reverse a condition's true/false value.

- Logical operators are primarily used with Boolean values, returning a Boolean result.
- While logical operators are mainly used with Boolean values, they can operate on any data type. Non-Boolean values are converted to Boolean:

  "falsy" values convert to false, and

  "truthy" values convert to true.

JavaScript

Operators

Logical Operators

- JavaScript recognizes six falsy values:

  false, null, undefined, 0, "" (empty string), and NaN.
  All other values, such as non-zero numbers and non-empty strings, are considered truthy.

Operator	Name	Description
&&	Logical AND	Returns true if both conditions are true; otherwise, returns false. It evaluates the operands from left to right. If the first operand is falsy, it will return the value of first operand, otherwise it will return the value of the second operand.
||	Logical OR	Returns true if at least one condition is true; otherwise, returns false. It also evaluates the operands from left to right. If the first operand is truthy, it will return the value of first operand, otherwise it will return the value of the second operand.
!	Logical NOT	Reverses the condition's value, turning true to false and false to true.It is a unary operator. It returns false if the operand can be converted to true, otherwise it returns true.

JavaScript

Operators

Assignment Operators

The assignment operators in JavaScript are used to assign values to the variables. These are binary operators. An assignment operator takes two operands, assigns a value to the left operand based on the value of right operand.

Operator	Name	Description
=	Assignment	Assigns the value on the right to the variable on the left.
+=	Addition Assignment	Adds the right value to the left variable and assigns the result to the variable.
-=	Subtraction Assignment	Subtracts the right value from the left variable and assigns the result to the variable.
*=	Multiplication Assignment	Multiplies the left variable by the right value and assigns the result to the variable.
/=	Division Assignment	Divides the left variable by the right value and assigns the result to the variable.
%=	Remainder Assignment	Assigns the remainder of dividing the left variable by the right value to the variable.
**=	Exponentiation Assignment	Raises the left variable to the power of the right value and assigns the result to the variable.

JavaScript

Operators

Bitwise Operators

The bitwise operators in JavaScript perform operations on the integer values at the binary level. Bitwise operators are similar to logical operators but they work on individual bits. Bitwise operators work on 32-bits operands.

Operator	Name	Description
&	Bitwise AND	Returns 1 if both bits are 1, otherwise 0.
|	Bitwise OR	Returns 1 if either bit is 1, otherwise 0.
^	Bitwise XOR	Returns 1 if both bits are different, otherwise 0.
~	Bitwise NOT	Returns 1 if bit is 0, otherwise 0.
<<	Left Shift	Shifts the bits left by pushing zeros in from right and discarding leftmost bits.
>>	Right Shift	Shifts the bits right by pushing copies of leftmost bit in from left and discarding rightmost bits.
>>>	Right Shift with Zero	Shifts the bits right by pushing zeros in from left and discarding rightmost bits.

JavaScript

Operators

Conditional Operator (? :)

The conditional operator in JavaScript first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation. The conditional operator is also known as the ternary operator.

Syntax condition ? exp1 : exp2;

– If is condition true, exp1 is executed else exp2 is executed.

Example:

```
let num1=30;
let num2=4;
document.write(num1>num2 ? num1 : num2);
```

JavaScript

Operators

Nullish coalescing operator (??)

It returns the first operand if it’s not null/undefined. Otherwise, the second one.

Syntax operand1 ?? operand2;

Example 1:

```
let user;
alert(user ?? "Anonymous"); // Anonymous (user is undefined)
```

Example 1:

```
let user = "John";
alert(user ?? "Anonymous"); // John (user is not null/undefined)
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-4

Today's Target

- JavaScript
  - Statements
    - Sequential Statements
    - Conditional (Alternating) Statements
    - Iteration (Looping) Statements
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Statements

In JavaScript, statements are instructions that tell what actions to perform.

- ➢ They form the building blocks of any JavaScript program, guiding the flow and logic of code.
- ➢ Each statement executes a specific task, such as performing calculations, controlling the sequence of code execution, defining variables, or handling decision-making processes.
- ➢ Statements are executed in a top-down sequence unless specifically directed otherwise by control flow commands like loops or conditionals.
- ➢ Based on how control flows through various statements in a script, statements can be categorized into three types:

➢ Sequential:	Statements that execute in sequence, one after another.
➢ Conditional (Alternating):	Statements that create decision-making in the code.
➢ Iteration (Looping):	Statements that repeat a block of code based on a condition.

JavaScript

Statements

Sequential Statements

- Sequential statements execute one after another in the order they appear in the code.
- These statements are the simplest and most fundamental type in programming, where each line runs only once from top to bottom unless interrupted by other control flow statements.
- Multiple statements may be grouped together in a block of statements using { }.

- Declaration Statements:

  Define variables and constants that hold values or references.

  Example
  ```
  var name = “Gateway";        // Declares a variable using 'var’
  let age = 25;                // Declares a variable using 'let’
  const PI = 3.14;             // Declares a constant using 'const’
  ```
- Expressions

  Perform calculations, assignments, or call functions.

  Example
  ```
  let x = 5 + 10;              // Addition and assignment
  document.write(x);           / Function call
  ```

JavaScript

Statements

Conditional (Alternating) Statements

Conditional statements are used when we need to perform different actions based on different conditions.

- Conditional statements allow the execution of different blocks of code based on certain conditions.
- They help control the flow of a program by making decisions.

- ‘if’ statement:

  The if(...) statement evaluates a condition in parentheses and, if the result is true, executes a block of code.

  Example

```
let percentAtt = 76;
if (percentAtt >= 75) {
        document.write(“You are allowed to appear in exams.");
}
```

JavaScript

Statements

Conditional (Alternating) Statements

- ‘if...else’ Statement:

  The if statement may contain an optional else block. It executes when the if condition is false.

  Example

```
let percentAtt = 76;
if (percentAtt >= 75) {
        document.write(“You are allowed to appear in exams.");
}
else {
        document.write(“You are not allowed to appear in exams.");
}
```

JavaScript

Statements

Conditional (Alternating) Statements

- ‘else...if’ Statement (several conditions):

  ‘else...if’ statement is used to test several variants of a condition.

Example

```
let num1=20;
let num2=10;
if(num1>num2) {
            document.write(num1+ " is greter than "+num2);
}
else if(num1==num2) {
             document.write(num1+ " is equal to "+num2);
}
else {
             document.write(num1+ " is less than "+num2);
}
```

JavaScript

Statements

Conditional (Alternating) Statements

- ‘switch’ Statement:

  switch statement is used when one of many blocks of code is to be selected for execution.
  - A switch statement can replace multiple if checks.
  - It gives a more descriptive way to compare a value with multiple variants.

```
switch(x) {
  case 'value1':
    ...
    break;
  case 'value2’:
    ...
    break;
  default:
    ...
    break;
}
```

The value of x is checked for a strict equality to the value from the first case (that is, value1) then to the second (value2) and so on.

If the equality is found, switch starts to execute the code starting from the corresponding case, until the nearest break (or until the end of switch).

If no case is matched then the default code is executed (if it exists).

JavaScript

Statements

Conditional (Alternating) Statements

➢ ‘switch’ Statement:

Example

```
let day = 3;
switch(day)
{
    case 1:   document.write("Monday");                break;
    case 2:   document.write("Tuesday");               break;
    case 3:   document.write("Wednesday");             break;
    case 4:   document.write("Thursday");              break;
    case 5:   document.write("Friday");                break;
    case 6:   document.write("Saturday");              break;
    case 7:   document.write("Sunday");                break;
    default:   document.write("Invalid value for day!");
}
```

JavaScript

Statements

Iteration (Looping) Statements

These statements are used to repeat the same code multiple times.

➢ ‘while’ loop

While the condition is true, the code from the loop body is executed.

- Along with comparisons, any expression or variable can be a loop condition: the condition is evaluated and converted to a Boolean by while.
- It is recommended to design the loop in a way that the condition will eventually become false after a specific number of iterations. This helps to avoid infinite loops.

Example

```
let i = 0;
while (i < 3) // loop condition, may also be written as while (i)
{
        document.write( i );
        i++;                    // will lead the loop condition to become false after 3 iterations.
}
```

JavaScript

Statements

Iteration (Looping) Statements

➢ ‘do .. while’ loop

In this type of loop, the condition check is moved below the loop body.

- The loop will first execute the body, then check the condition, and, while it’s true, execute the loop body over and over until the condition becomes false.
- This loop should only be used when you want the body of the loop to execute at least once regardless of the condition being true/ false.

Example

```
let i = 0;
do {
        document.write( i );
        i++;
} while (i < 3);
```

JavaScript

Statements

Iteration (Looping) Statements

- ‘for’ loop

  This loop, repeatedly executes a block of code a specified number of times, based on an initialization, a condition to evaluate at each iteration, and a step to update the loop counter.

  Example

```
for (let i = 3; i > 0; i--) {
        alert(i);
}
```

initialization	*let i = 3*	*Executes once upon entering the loop.*
condition	*i > 0*	*Checked before every loop iteration. If false, the loop stops.*
body	*alert(i)*	*Runs again and again while the condition is truthy.*
step	*i--*	*Executes after the body on each iteration.*

Note: Any part of for can be skipped.

JavaScript

Statements

'for..in' and 'for..of' loops

- 'for..in' Statement

  It is used to iterate through all the properties (keys) of an object.
- 'for..of' Statement

  It is used to iterate through all the elements of an array.

Example 1

```
 let course= { name: "Web Technology",
code: "BCS 502" };

for (let property in course) {
            document.write(property);
            document.write(course[property]);
}
```

Example 2

```
let courses = ["WT", "DAA", "DBMS"];


for (let course of courses) {
          document.write(course);
}
```

JavaScript

Statements

‘break’ and ‘continue’ Statements

- ‘break’ Statement

  It is used to exit from the current loop, switch statement, or labeled block.
- ‘continue’ Statement

  It is used to skip the current iteration of a loop and continue with the next iteration.

Example 1

```
for (let i = 0; i < 5; i++) {
    if (i === 3) {
        break;  // Exits the loop when i is 3
    }
    document .write(i);
}
// Output: 0 1 2
```

Example 2

```
for (let i = 0; i < 5; i++) {
    if (i === 3) {
        continue;  // Skips the iteration when i is 3
    }
     document .write(i);
}
// Output: 0 1 2 4
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-5

Today's Target

- JavaScript
  - Functions
    - Function Declaration
    - Returning values
    - Default values for parameters
    - Function Expressions
    - Arrow Functions
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Functions

Functions are the main "building blocks" of the program. Functions allow to perform a similar action in many places of the script.
They allow the code to be called many times without repetition i.e. they avoid code duplication.

Function Declaration

Function declaration is used to create a function in JavaScript. In a function declaration,

- First we write the function keyword then
- the name of the function, after it
- a list of parameters between the parentheses (comma-separated, or empty) and finally
- the code of the function, also named "the function body", between curly braces.

Examples:

```
function show() {
 document.write("Web Technology");
}
show();
```

```
function showCourse(course) {
        document.write(course);
}
showCourse("DBMS");
```

JavaScript

Functions

- A *local variable* is one that is declared inside a function. It is only visible inside that function.
- An *outer variable* is one that is declared outside a function. A function can access an outer variable as well. Variables declared outside of any function, are called global variables.
- If a same-named variable is declared inside the function then it shadows the outer one.
- Global variables are visible from any function (unless shadowed by locals).

Example:

```
let outerGlobal = 10;
let inner=20;
function print1() {
        inner=30;
        document.write(outerGlobal); //Output: 10
        outerGlobal=40;
        document.write(inner);
        //Output: 30, local variable shadowed outer.
}
print1();
```

```
function print2() {
        document.write(outerGlobal);
        //Output: 40, print1() changes it to 40
}       print2();
```

JavaScript

Functions

Returning a value from function

A function can return a value back into the calling code.

Example:

```
function add(a, b){
        return a+b;
}
document.write(add(5,6))        //Output: 11
```

- ➢ The ‘return’ can be in any place of the function. When the execution reaches it, the function stops, and the value is returned to the calling code.
- ➢ There may be many occurrences of return in a single function.
- ➢ It is possible to use return without a value. That causes the function to exit immediately.
- ➢ A function with an empty return or without it returns undefined
- ➢ Never add a newline between return and the value. It will not work, because JavaScript will assume a semicolon after return in this case.

JavaScript

Functions

Default values for parameters

- If a function is called, but an argument is not provided, then the corresponding value becomes undefined.
- We can specify our own default values for parameters in the function declaration, using =.

Example:

```
function add(a, b=5){
        return a+b;
}
document.write(add(5,6))          //Output: 11, as a=5, and b=6
document.write(add(5))   //Output: 10, as a=5, and b=5
document.write(add())    //Output:NaN, as a=undefined, and b=5
```

- All the default values must be specified at the end.

Example:

```
function add(a=10, b){          return a + b;      }
add(,10); // Error              add(20); // NaN
```

JavaScript

Functions

Function Expressions

If the function is created as a part of an expression, it's called a "Function Expression". Functions are values in JavaScript. They can be declared, assigned, or copied in any place of the code.

Examples:

```
//Function Declaration
function namaskar() {
        document.write("Namaste");
}
```

```
//Function Expression
let welcome = function() {
        document.write("Welcome");
};
```

- In function expression, there is no name after the function keyword, so this is an *anonymous function*. As we immediately assign it to the variable, thus it will be identified with the name of the variable: *welcome*.
- No matter how the function is created, a function is a value. Both examples above store a function in the variables '*namaskar*' and '*welcome*' respectively.

JavaScript

Functions

Function Expressions

As the functions are values in JavaScript, so the following line will write the code of the function as output.

```
document.write(welcome); //Writes function's code
```

A function is a special value, in the sense that we can call it as following. Mention the parenthesis at the end.

```
welcome(); // Calls function
```

A function value may be copied to another variable as written in the following:

```
welcome = namaskar;
```

A Function Expression is created when the execution reaches it and is usable only from that moment whereas A Function Declaration can be called earlier than it is defined.

Example:

```
namaskar();  //Writes Namaste
//welcome(); //Error: Cannot access 'welcome' before initialization
//Function Expression
let welcome = function() { document.write("Welcome");          };
// Function Declaration
function namaskar() {       document.write("Namaste");          }
```

JavaScript

Functions

Arrow Functions

Arrow functions is very simple and concise way for creating functions.

It's called "arrow function", because it has the following syntax:

let func = (arg1, arg2, ..., argN) => expression;

Arrow functions may be specified in two ways:

Without curly braces: (...args) => expression – the right side is an expression: the function evaluates it and returns the result.

With curly braces: (...args) => { body } – curly braces allow us to write multiple statements inside the function, but we need an explicit return to return something.

Examples:

```
let add = (a, b) => a + b;
```

```
let add = (a, b) => {          let result = a + b;
                               return result;
              };
```

- If we have only one argument, then parentheses around parameters can be omitted, making that even shorter. *Example:* let double = n => n * 2;
- If there are no arguments, parentheses are empty, but they must be present.

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-6

Today's Target

- JavaScript
  - Popup Dialog Boxes
    - alert
    - confirm
    - prompt
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Popup Dialog Boxes: alert, confirm, and prompt

In JavaScript, a popup dialog box is a small, temporary window that appears on the current web page to convey a message or interact with the user.

JavaScript provides three main types of popup dialog boxes: alert, confirm, and prompt.

- alert, confirm, and prompt are built-in *functions of the window object*, that are directly accessible in the script.
- Each dialog box *demands user action*, making them effective for obtaining input, confirmations, or alerting users before proceeding with further code.
- These dialog boxes *temporarily halt execution of the script* until the user interacts with them, ensuring any subsequent actions wait for user input.

JavaScript

Popup Dialog Boxes: alert, confirm, and prompt

'alert' box

The alert function shows a message and waits for the user to press "OK".

Example: alert("Web Technology");

'confirm' box

The function confirm shows a window with a question and two buttons: OK and Cancel.

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Example: confirm("Do you want to delete the file?");

JavaScript

Popup Dialog Boxes: alert, confirm, and prompt

'prompt' box

The prompt function is used when we want the user to input a value.

It shows a window with

- a text message,
- an input field for the user, and
- the buttons OK and Cancel.
- *If the user clicks "OK" the box returns the input value. If the user clicks "Cancel", the box returns null.*

Example: prompt("Input a number", 2);

The function prompt accepts two arguments:
First parameter is the text to show to the user and an *optional* second parameter, the initial value for the input field.

JavaScript

***Popup Dialog Boxes*: alert, confirm, and prompt**

Example

```
<script>
            let num = prompt("Input a number");
           document.write(typeof num)
           let isConfirm = confirm("Your input is "+num);
           if(isConfirm)
                    alert("The double of "+num+" is "+num*2);
</script>
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-7

Today's Target

- JavaScript
  - Arrays
    - Creating and accessing elements
    - 'length' property and Array Methods
    - Iterating through Arrays
    - Multidimensional Arrays
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Arrays

An array is a special type of object in JavaScript that can hold multiple values in a single container.

Arrays allow us to store a collection of elements under a single name.

- Arrays are *objects with indexed elements*, making them different from traditional arrays in other languages.
- Elements are *ordered and indexed*, starting at 0.
- Arrays can store *different types of data* in each element (e.g., numbers, strings, objects).
- They are *dynamic*, meaning their size can grow or shrink as needed.

JavaScript

Arrays

Creating an Array

Using *square brackets*

Example: `let fruits = ["apple", "banana", "cherry"];`

Using *new Array()*

Example:

```
let numbers = new Array(1, 2, 3);
let numbers = new Array(5); // Creates an empty array of size 5.
```

Accessing elements from an Array

Elements are accessed from an array using their respective indexes.

Example: `document.write(fruits[1]) // Output: banana`

JavaScript

Arrays

'length' property

length: It gives the number of elements in the array.

Example:

```
let arr = [10, 20, 30];
console.log(arr.length); // Output: 3
```

Array Methods

push(): Adds one or more elements to the end of an array and returns the new length.

Example: `arr.push(40); // Adds 40`

pop(): Removes the last element of the array and returns it.

Example: `arr.pop(); // Removes 40`

JavaScript

Arrays

Array Methods

shift(): Removes the first element of the array and returns it.

Example: `arr.shift(); // Removes 10`

unshift(): Adds one or more elements to the beginning of an array and returns the new length.

Example: `arr.unshift(5); // Adds 5`

indexOf(): Returns the first index of a specified value in the array, or -1 if not found.

Example: `arr.indexOf(20); // Output: 1`

includes(): Checks if the array contains a specified value and returns true or false.

Example: `arr.includes(30); // Output: true`

JavaScript

Arrays

Array Methods

splice(): Adds or removes elements at a specific index.
Example: `arr.splice(1, 2); // Removes 2 elements from index 1`

slice(): Returns a portion of the array from start to end-1 indexes.
Example: `let newArr = arr.slice(0, 2); // Copies first two elements`

concat(): Merges two or more arrays and returns a new array.
Example: `let combined = arr.concat([40, 50]); // Combines arrays`

join(): Joins all array elements into a string separated by a specified delimiter.
Example: `arr.join("-"); // Output: "10-20-30"`

JavaScript

Arrays

Array Methods

reverse(): Reverses the order of elements in the array.

Example: arr.reverse(); // [30, 20, 10]

sort(): Sorts the elements of the array in place and returns the sorted array.

Example: arr.sort(); // Default: Lexicographical sorting, elements are sorted as strings.

To use our own sorting order, we need to supply a function as the argument of sort().

Example: *arr.sort((a,b)=>a-b); //Soring will be done as per the comparison function*

map(): Creates a new array with the results of calling a function on every element.

Example: let squared = arr.map(x => x * x); // Squares each element

forEach(): Executes a function on each array element without returning a new array, unlike map(), which returns a new transformed array.

Example: arr.forEach(num => document.write(num * 2)) //Writes double of each array element on the document.;

JavaScript

Arrays

Array Methods

filter(): Creates a new array with all elements that pass a test implemented by a function.

Example: `let even = arr.filter(x => x % 2 === 0); // Filters even numbers`

reduce(): Executes a reducer function on each element and returns a single result.

Example: `let sum = arr.reduce((a, b) => a + b, 0); // Sums elements`

find(): Returns the first element that satisfies a given condition.

Example: `let found = arr.find(x => x > 15); // Finds first element > 15`

isArray(): Checks if the given value is an array.

Example: `Array.isArray(arr); // Output: true`

JavaScript

Arrays

Array Methods

every(): Checks if all elements pass a test implemented by a function.

Example: `arr.every(x => x > 5); // true if all elements > 5`

at(): allows to access elements in an array using both positive and negative indexes, where negative indexes count from the end of the array.

Example: `arr.at(-2); // returns second last element from the array.`

toString(): Converts the array to a comma-separated string.

Example: `arr.toString(); // 10,20,30`

Note: When an array is passed to a method like document.write() in JavaScript, it is automatically converted to a string. JavaScript calls the array's toString() method, which joins all elements in the array into a single string, separated by commas.

Example: `document.write(arr);       // Output: 10,20,30`

JavaScript

Arrays

Iterating through Arrays

Iterating through an array means accessing and processing each element of the array, one by one, in a sequential manner.

Arrays in JavaScript are ordered collections that can be iterated using various methods.

for loop

Allows index-based iteration.

Example:

```
const numbers = [1, 2, 3, 4];
for (let i = 0; i < numbers.length; i++) {    console.log(numbers[i]);    }
```

for...of loop

Allows simple iteration over array elements. It iterates over the values of an array, not the indexes, making it suitable for when you only need to access the elements, not their positions.

Example:

```
for (let num of numbers) {    document.write(num);    }
```

JavaScript

Arrays

Multidimensional Arrays

An array of arrays, where each element of the array is itself an array.

Example

```
let matrix = [ [1, 2, 3],  [4, 5, 6],  [7, 8, 9] ];
// Accessing elements
document.write(matrix[1][2]); // Outputs 6 (second row, third column)
```

Iterating through Multidimensional Arrays:

We need to iterate through each dimension of the array.

```
for (let row of matrix) {
        for (let col of row) {
                document.write(col);
        }
}
```

Note: We can create arrays with more than two dimensions as well in the same way. (3D arrays, etc.).

AKTU

B.Tech 5th Sem

GW
GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-8

Today's Target
- JavaScript
  - 'document' Object
    - Properties
    - Methods
    - Writing on the document
    - Manipulating DOM Elements
    - Creating, Adding, and Removing DOM Elements
  - AKTU PYQs

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

'document' Object

The document object in JavaScript provides access to all elements of an HTML document. It acts as an interface to the Document Object Model (DOM), allowing JavaScript to access and manipulate HTML document dynamically.

- When an HTML document is loaded into a web browser, it automatically creates a corresponding document object, which acts as the *root of the document's structure*.
- The document object includes various *properties and methods* that allow you to retrieve information about HTML elements and dynamically modify them.
- In JavaScript, the document object is a *property of the window object*. You can access it using the window.document syntax, but it is also accessible directly without prefixing it with the window object as window is a global object.

JavaScript

‘document’ Object

Properties

document.documentElement

Refers to the <html> element of the document and used to access or manipulate the root element of the document.

Example: document.write(*document.documentElement*);

document.head

Refers to the <head> element of the document and allows access to or manipulation of the <head> section of the HTML.

Example: document.write(*document.head*);

document.title

Represents the title of the HTML document and used to get or set the title using this property.

Example: document.write(*document.title*);
document.title = “Gateway Classes"; // Changes the title

JavaScript

‘document’ Object

Properties

document.body

Refers to the <body> element of the document and allows access to or modification of the content within the <body> tag.

Example: document.write(*document.body*);

document.URL

Provides the complete URL of the document currently loaded.

Example: document.write(*document.URL*);

JavaScript

‘document’ Object

Methods

document.getElementById(id)

Returns a single element with the specified value of the id attribute. It is used to access a unique element directly.

Example: `let element = document.getElementById("myId");`

document.getElementsByClassName(className)

Returns a live HTMLCollection of elements with the specified class name. It is used to access multiple elements sharing the same class.

Example: `let elements = document.getElementsByClassName("myClass");`

JavaScript

‘document’ Object

Methods

document.getElementsByTagName(tagName)
Returns a live HTML collection of elements with the specified tag name. It is used to access all elements of a specific type, e.g., all <div> or <p> elements.

Example: `let paragraphs = document.getElementsByTagName("p");`

document.getElementsByName(name)
Returns all the elements with the specified value of the name attribute. It is commonly used for form elements like <input> or <select>.

Example: `let inputs = document.getElementsByName("username");`

Note: All these methods (except getElementsByName) return live collections, meaning they update automatically if the DOM changes.

JavaScript

‘document’ Object

Methods

querySelector(selector)

Selects the first element in the DOM that matches the specified CSS selector.

Example: let firstItem = document.querySelector("li.active");

querySelectorAll(selector)

Selects all elements in the DOM that match the specified CSS selector

Example: let allButtons = document.querySelectorAll("p");

Note:

- ➢ Both querySelector and querySelectorAll are versatile methods that provide a modern and powerful way to access elements compared to traditional getElementBy... methods.
- ➢ They support any valid CSS selectors like .class, #id, [attribute], pseudo-classes, etc.

JavaScript

'document' Object

Writing on the document

document.write():

Writes the given HTML content or text directly to the document.

document.writeln():

Writes the given HTML content or text directly to the document and adds a new line at the end.

Example:

```
document.write("<pre>")
        document.write("Hello")
        document.write("World")
document.write("</pre>")
```

```
document.write("<pre>")
        document.writeln("Hello")
        document.writeln("World")
document.write("</pre>")
```

Browsers treat multiple spaces or newline characters in HTML as a single space when rendering content. This is why the \n from writeln() doesn't visibly change the layout. The <pre> tag, used above in the example, preserves spaces and newlines in the output, making the effect of writeln() clear.

JavaScript

‘document’ Object

Manipulating DOM Elements

Changing Content

innerHTML:

This property retrieves or sets the HTML content (including tags) of an element.

Example: let element=document.getElementById(“para1”);
element.innerHTML = "<b>Hello World!</b>";

textContent:

This property retrieves or sets the plain text content of an element without any tags.

Example: element.textContent = "Hello World!";

Changing Styles

style:

This property is used to access or set inline CSS styles directly on an element.

Example: element.style.backgroundColor = "blue";

JavaScript

'document' Object

Manipulating DOM Elements

Changing Attributes

setAttribute:

Sets the value of a specified attribute on an element.

Example:

```
let element = document.getElementsByTagName("img")[0]
element.setAttribute("src", "image.jpg");
```

getAttribute:

Retrieves the value of a specified attribute on an element.

Example:

```
element.getAttribute("src");
```

JavaScript

'document' Object

Creating, Adding, and Removing Elements in the DOM

Creating and Adding Elements to the DOM

createElement():

Creates a new HTML element.

Example: `let newDiv = document.createElement("div");`

appendChild():

Adds a new child element to the end of a parent element.

Example: `parentElement.appendChild(newDiv);`

insertBefore():

Inserts a new element before a specified child element.

Example: `parentElement.insertBefore(newDiv, existingChild);`

JavaScript

'document' Object

Creating, Adding, and Removing Elements in the DOM

Removing Elements from the DOM

removeChild():

Removes a child element from its parent.

Example: `parentElement.removeChild(childElement);`

replaceChild():

Replaces an existing child element with a new one.

Example: `parentElement.replaceChild(newDiv, oldChild);`

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-9

Today's Target

- JavaScript
  - Forms
    - Accessing form & its elements
    - Using properties of form elements
    - Form Events
    - Form validation
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

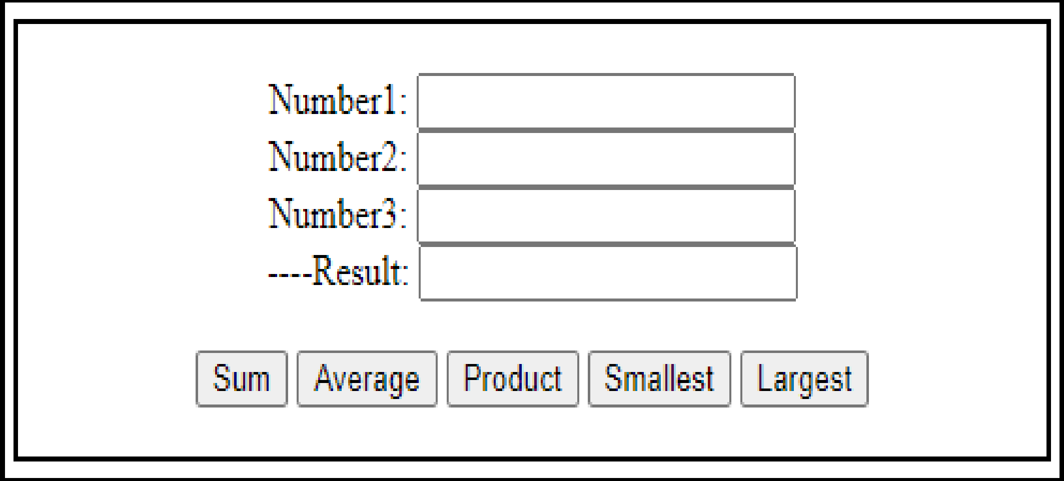
Forms

Forms enable the users to interact with the web page dynamically and/or to submit data to the server for processing.

Role of JavaScript in Forms

The use of JavaScript with Forms enhances interactivity and responsiveness by:

- Accessing and manipulating form elements.
- Validating input before submission.
- Handling form events dynamically.



JavaScript

Forms

Accessing form & its elements in JavaScript

We access form elements in JavaScript to dynamically manipulate, validate, or retrieve input values from forms.

Accessing Forms

We may access forms by their index or name.

Examples:

```
let form = document.forms.myForm        // Uses Name
let form = document.forms[0]            // Uses Index
```

Accessing Elements of Form

We may access form elements by their name or id.

Examples:

```
let number1 = form.elements.number1;        let number1 = form.elements["number1"]
alert(number1[0].value);                    alert(number1[0].value);

let number1 = form.number1
alert(number1[0].value);
```

JavaScript

Forms

Accessing form & its elements in JavaScript

Accessing Elements of Form

Examples:

```
let number1 = document.getElementById("num1")
alert(number1.value);

let number1 = document.getElementsByName("number1")
alert(number1[0].value);

let number1 = document.querySelector("#num1")
alert(number1.value);

let number1 = document.querySelectorAll("#num1")
alert(number1[0].value);
```

JavaScript

Forms

Using 'Properties' of form elements in JavaScript

S.No.	Property	Description	Example
1	value	Gets or sets the current value of an input field.	let name = document.getElementById("username").value;
2	checked	Indicates whether a checkbox or radio button is selected.	let isChecked = document.getElementById("agree").checked;
3	disabled	Specifies if the element is disabled or not.	document.getElementById("submit").disabled = true;
4	readOnly	Specifies whether the user can edit the input field.	document.getElementById("username").readOnly = true;
5	type	Retrieves the type of input element (e.g., text, password, checkbox).	let inputType = document.getElementById("password").type;
6	name	Returns or sets the name attribute of the form element.	let fieldName = document.getElementById("email").name;
7	placeholder	Displays a hint or placeholder text in the input field.	document.getElementById("username").placeholder = "Enter your name";
8	selectedIndex	For dropdowns, retrieves the index of the selected option.	let index = document.getElementById("dropdown").selectedIndex;

JavaScript

Forms

Form Events in JavaScript

Form events in JavaScript are triggers that occur when users interact with form elements.

- ➢ The events allow developers to handle user actions dynamically, such as submitting a form, changing the value of an input field, or clicking a button.
- ➢ The event handlers can be used to execute specific code when an event occurs, such as validating input, manipulating DOM elements
- ➢ JavaScript provides various event handlers to respond to these events, enabling real-time validation, feedback, and interactive behavior without needing to refresh the page.
- ➢ JavaScript provides event handlers that can be attached to form elements using attributes like onsubmit, onchange, onclick, and more.
- ➢ By using these events, developers can enhance the user experience and control form interactions effectively.

JavaScript

Forms

Form Events in JavaScript

S.No.	Event Handler	Description	Example
1	onsubmit	Triggered when the form is submitted.	<form onsubmit="return validateForm()">Submit</form>
2	onchange	Fired when the value of an element changes.	<input type="text" onchange="alert('Changed!')">
3	onfocus	Fired when an element gains focus.	<input type="text" onfocus="this.style.backgroundColor='yellow'">
4	onblur	Fired when an element loses focus.	<input type="text" onblur="this.style.backgroundColor='white'">
5	onclick	Triggered when the user clicks on an element.	<button onclick="alert('Button Clicked!')">Click Me</button>
6	oninput	Fired when the user types in an input field.	<input type="text" oninput="document.getElementById('demo').innerHTML = this.value;">
7	onreset	Fired when the form is reset.	<form onreset="alert('Form reset!')"><input type="reset"></form>
8	onselect	Triggered when the user selects text in an input field.	<textarea onselect="alert('Text Selected')">Select some text</textarea>
9	onkeypress	Fired when a key is pressed down in an input field.	<input type="text" onkeypress="alert('Key Pressed!')">

JavaScript

Forms

Form validation using JavaScript

Form validation is a process of ensuring that the data entered by a user into a form meets the required criteria before it is submitted.

JavaScript for validating forms:

JavaScript plays a key role in form validation by providing a dynamic, interactive, and real-time mechanism to check and enforce input requirements.

- Real-Time Feedback: Users receive instant notifications for invalid inputs, improving user experience.
- Efficiency: Prevents invalid data from being sent to the server, saving server processing time and bandwidth.
- Custom Validation Rules: Allows developers to implement specific rules for fields, such as regex for patterns or range checks for numbers.

JavaScript

Forms

Form validation using JavaScript

Types of Form Validation with JavaScript

➢ Required Field Validation:	Ensures critical fields are not left blank.
➢ Format Validation:	Validates inputs like email, phone numbers, and URLs that they follow specified format
➢ Range Validation:	Confirms numeric values or dates fall within an acceptable range.
➢ Cross-Field Validation:	Compares multiple fields (e.g., password and confirm password).
➢ Custom Validation:	Applies specific, custom rules defined by the application.

JavaScript

Forms

Form validation using JavaScript

Example:

```
function validateForm() {
    let name = document.getElementById("name").value;
    if (name === "") {
        alert("Name cannot be empty.");
        return false;
    }

    let age = document.getElementById("age").value;
    if (isNaN(age) || age < 1 || age > 120) {
        alert("Please enter a valid age.");
        return false;
    }
    return true;
}
```

JavaScript

Forms

Form validation using JavaScript

Design a HTML form with name, address, pincode and password. Outline a JavaScript code that can validate on following constraints:

a. No fields must be left blank

b. Pincode must be numeric and contain 6 digits

c. Password field

(i) At least contains 1 Capital letter.

(ii) Minimum length 8 characters.

```
// a. No fields must be left blank
   if (!name || !address || !pincode || !password) {
      alert("All fields are required.");
      return false;
   }
```

```
// b. Validate Pincode
   if (isNaN(pincode) || pincode.length !== 6) {
      alert("Pincode must be a 6-digit numeric value.");
      document.getElementById("pincode").focus();
      return false;
   }
```

JavaScript

Forms

Form validation using JavaScript

```
// c. Validate Password Length
  if (password.length < 8) {
    alert("Password must be at least 8 characters long.");
    document.getElementById("password").focus();
    return false;
  }
```

```
// c. Validate Password to Ensure It Contains at Least One Capital
        Letter
  let hasUppercase = false;
  for (let i = 0; i < password.length; i++) {
    if (password[i] >= 'A' && password[i] <= 'Z') {
      hasUppercase = true;
      break;
    }
  }

  if (!hasUppercase) {
    alert("Password must contain at least one uppercase letter.");
    document.getElementById("password").focus();
    return false;
  }
```

AKTU

B.Tech 5th Sem

GW
GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-10

Today's Target

- JavaScript
  - Objects
    - Working with Objects
    - Object Methods
    - 'this' Keyword
    - Creating Objects with Constructors
    - Iterating Through Object
    - Useful Built-In Object Methods
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

Objects

Objects are collections of key-value pairs in JavaScript.

- ➢ They are used to represent real-world entities (e.g., a student, projector, or fan).
- ➢ They are used to store multiple related pieces of data in one place.
- ➢ They are used to organize and group data.

Characteristics of Objects:

- ➢ Objects have properties. Keys in the key-value pairs are called properties.
- ➢ Values can be any data type (numbers, strings, arrays, functions, or even other objects).
- ➢ Objects are mutable — you can add, change, or remove properties.

JavaScript

Objects

Syntax

```
let objectName = {
    key1: value1,
    key2: value2,
    key3: value3
};
```

Example

```
let person = {
    name: “Anubhav",
    age: 18,
    isStudent: true
};
```

JavaScript

Objects

Working with Objects

Accessing Object Properties:

Dot Notation: Used to access properties directly by their name.

Example: `console.log(person.name);`

Square Bracket Notation: Used to access properties using name and also dynamically using variables or expressions.

Example: `console.log(person["age"]);`

Modifying Properties:

A new property can be added to an object or an existing property can be updated.

Example:

```
person.isStudent = false;        // Update
person.city = "New York";        // Add
```

Deleting Properties:

A property that is no longer required can be deleted.

Example: `delete person.age;`

JavaScript

Objects

Object Methods

A method is a function defined inside an object.
In JavaScript, functions are treated as values, means that they can be assigned to object properties just like other built-in type values.

Example:

```
let user = {
    name: "John",
    greet: function() {
        console.log("Hello");
    }
};
```

JavaScript

Objects

'this' Keyword

It refers to the current object where the method is called.

Example:

```
let user = {
    name: "Anubhav",
    greet: function() {
        console.log("Hello, " + this.name);
    }
};

user.greet(); // Output: Hello, Anubhav
```

JavaScript

Objects

Creating Objects with Constructors

A special function used to create multiple objects with the same structure.
The regular {...} syntax allows us to create one object. But often we need to create many similar objects, like multiple persons, books and so on. That can be done using constructor functions and the "new" operator.

Example:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

let person1 = new Person(“Samar", 25);
let person2 = new Person(“Digvijay", 30);

console.log(person1.name);          // Samar
```

When a constructor function is executed with new, it does the following steps:

1. It *creates empty object* and assigns to this.
2. The function body executes. Usually it modifies this, *adds new properties* to it.
3. It *returns the value of this*.

JavaScript

Objects

Iterating Through Object

For...in loop is used to iterate over all keys in an object.

Example:

```
let book = {
    title: "JavaScript Basics",
    author: "Alice",
    year: 2021
};

for (let property in book) {
    console.log(property + ": " + book[property]);
}
```

JavaScript

Objects

Useful Built-In Object Methods

Object.keys()

Returns an array of keys in the object.

Example:

```
let car = { brand: "Tesla", model: "X" };
console.log(Object.keys(car));                    // ["brand", "model"]
```

Object.values()

Returns an array of values in the object.

Example:

```
console.log(Object.values(car));                  // ["Tesla", "X"]
```

Object.entries()

Returns an array of [key, value] pairs.

Example:

```
console.log(Object.entries(car));
                         // [["brand", "Tesla"], ["model", "X"]]
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-11

Today's Target
- JavaScript
  - Objects
    - String Object
    - Math Object
    - Date Object
  - AKTU PYQs

By Amol Sharma sir
- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

'String' Objects

The String Object allows manipulation of text data.

- Strings are sequences of characters enclosed in quotes (single, double), or backticks (', ", or `).
- String literals are automatically treated as String objects when methods are invoked.
- Strings are immutable.

Constructing String Objects

String objects can be created using the String() constructor:

Example:

```
let strObj = new String("Hello");
console.log(typeof strObj); // object
```

Note: Use string literals ("text") for efficiency unless object behavior is specifically required.

JavaScript

'String' Objects

Common Property

Property	Description	Example
length	Returns the number of characters in a string.	"Hello".length → 5

Common Methods

Method	Description	Example
charAt(index)	Returns the character at the specified index.	"Hello".charAt(1) → e
toUpperCase()	Converts the string to uppercase.	"hello".toUpperCase() → HELLO
toLowerCase()	Converts the string to lowercase.	"HELLO".toLowerCase() → hello
substring(start, end)	Extracts a portion of the string.	"Hello".substring(1, 4) → ell
concat()	Combines strings.	"Hello".concat(" World") → Hello World
trim()	Removes whitespace from both ends of a string.	" Hello ".trim() → Hello
includes()	Checks if a string contains a specified value.	"Hello".includes("ll") → true

JavaScript

'Math' Object

The Math Object provides mathematical constants and functions.

- It does not have a constructor; you cannot create instances of Math.
- All methods and properties are accessed directly from Math (e.g., Math.sqrt(16)).

Common Properties

Property	Description	Example
Math.PI	The value of π (pi).	Math.PI → 3.14159
Math.E	The base of natural logarithms (Euler's number).	Math.E → 2.71828
Math.SQRT2	The square root of 2.	Math.SQRT2 → 1.414
Math.SQRT1_2	The square root of 1/2.	Math.SQRT1_2 → 0.707
Math.LN2	The natural logarithm of 2.	Math.LN2 → 0.693
Math.LN10	The natural logarithm of 10.	Math.LN10 → 2.302
Math.LOG2E	The base-2 logarithm of Euler's number.	Math.LOG2E → 1.442
Math.LOG10E	The base-10 logarithm of Euler's number.	Math.LOG10E → 0.434

JavaScript

'Math' Object

Common Methods

Method	Description	Example
Math.abs(x)	Returns the absolute value of x.	Math.abs(-7) → 7
Math.round(x)	Rounds x to the nearest integer.	Math.round(4.7) → 5
Math.ceil(x)	Rounds x up to the next largest integer.	Math.ceil(4.1) → 5
Math.floor(x)	Rounds x down to the next smallest integer.	Math.floor(4.9) → 4
Math.random()	Generates a random number between 0 and 1.	Math.random() → 0.6723
Math.max(a, b, ...)	Returns the largest value in a set of numbers.	Math.max(1, 5, 3) → 5
Math.min(a, b, ...)	Returns the smallest value in a set of numbers.	Math.min(1, 5, 3) → 1
Math.pow(base, exp)	Returns base raised to the power of exp.	Math.pow(2, 3) → 8
Math.sqrt(x)	Returns the square root of x.	Math.sqrt(16) → 4

JavaScript

‘Date’ Objects

The Date object in JavaScript is used to work with dates and times.

- It provides methods to manipulate date and time values.
- Date object internally represents time as the number of milliseconds since January 1, 1970 (UTC).

Constructors

Method	Description	Example
new Date()	Current date and time.	new Date()
new Date(dateString)	Parses a date string to create a Date object.	new Date("2024-11-17")
new Date(year, month, ...)	Constructs with individual components. year: Full year (e.g., 2024).monthIndex: Month index (0 for January, 11 for December). day: Day of the month (default: 1). hours, minutes, seconds, milliseconds: Optional; default to 0.	new Date(2024, 10, 17, 14, 30)
Date.now()	Returns milliseconds since epoch.	Date.now()
Date.parse(dateString)	Parses a date string and returns milliseconds since epoch.	Date.parse("2024-11-17T14:30:00")
new Date(milliseconds)	Constructs a date from milliseconds since epoch.	new Date(1732200000000)

JavaScript

‘Date’ Objects –

Common Methods

Method	Description	Example
getDate()	Returns the day of the month (1-31).	now.getDate() → 17
setDate()	Sets the day of the month (1-31).	now.setDate(25) → Updates to 25th
getMonth()	Returns the month (0-11).	now.getMonth() → 10 (November)
setMonth()	Sets the month (0-11).	now.setMonth(11) → Updates to December
getFullYear()	Returns the year.	now.getFullYear() → 2024
setFullYear()	Sets the year.	now.setFullYear(2025) → Updates to 2025
getDay()	Returns the day of the week (0-6).	now.getDay() → 5 (Friday)
getHours()	Returns the hour (0-23).	now.getHours() → 14
setHours()	Sets the hour (0-23).	now.setHours(10) → Updates to 10 AM

JavaScript

'Date' Objects

Common Methods

Method	Description	Example
getMinutes()	Returns the minutes (0-59).	now.getMinutes() → 45
setMinutes()	Sets the minutes (0-59).	now.setMinutes(30) → Updates to 30
getSeconds()	Returns the seconds (0-59).	now.getSeconds() → 30
setSeconds()	Sets the seconds (0-59).	now.setSeconds(15) → Updates to 15
getMilliseconds()	Returns the milliseconds (0-999).	now.getMilliseconds() → 500
setMilliseconds()	Sets the milliseconds (0-999).	now.setMilliseconds(250) → Updates to 250
getTime()	Returns the milliseconds since epoch.	now.getTime() → 1732190730000
setTime()	Sets the time in milliseconds since epoch.	now.setTime(1732200000000) → Updates to specified epoch time

AKTU PYQs

Web Technology (AKTU 2022 – 23)

Write a program in JavaScript to display digital clock showing HH:MM:SS.

```
<script>
    function updateClock() {
      var now = new Date();
      var hours = now.getHours();
      var minutes = now.getMinutes();
      var seconds = now.getSeconds();

      // Format time to always show 2 digits (e.g., 09 instead of 9)
      hours = (hours < 10) ? "0" + hours : hours;
      minutes = (minutes < 10) ? "0" + minutes : minutes;
      seconds = (seconds < 10) ? "0" + seconds : seconds;

      // Combine the time parts
      var time = hours + ":" + minutes + ":" + seconds;

      // Display the time in the HTML element with id 'clock'
      document.getElementById('clock').textContent = time;
    }
```

AKTU PYQs

Web Technology (AKTU 2022 – 23)

Write a program in JavaScript to display digital clock showing HH:MM:SS.

```
        // Update the clock every 1 second
        setInterval(updateClock, 1000);

        // Initialize the clock immediately
        updateClock();

    </script>
```

AKTU PYQs

Web Technology (AKTU 2021 – 22)

Design a JavaScript program to display the current day and time in the following format.
Today is : Monday.
Current time is : 11 AM : 50 : 58

```
<script>
    function displayDateAndTime() {
        var now = new Date();
        var dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
    "Saturday"];

        // Get current day
        var currentDay = dayNames[now.getDay()];

        // Get current time
        var hours = now.getHours();
        var minutes = now.getMinutes();
        var seconds = now.getSeconds();
```

AKTU PYQs

Web Technology (AKTU 2021 – 22)

Design a JavaScript program to display the current day and time in the following format.
Today is : Monday.
Current time is : 11 AM : 50 : 58

```
    // Format the hours to show AM/PM
    var period = (hours >= 12) ? "PM" : "AM";
    hours = (hours > 12) ? hours - 12 : hours;
    hours = (hours == 0) ? 12 : hours; // Adjust if hours is 0 (midnight)

    // Ensure minutes and seconds are two digits
    minutes = (minutes < 10) ? "0" + minutes : minutes;
    seconds = (seconds < 10) ? "0" + seconds : seconds;

    // Format time and display the output
    var timeString = "Today is : " + currentDay + ".<br>Current time is : " + hours + " " + period + " : "
+ minutes + " : " + seconds;
    document.getElementById('timeDisplay').innerHTML = timeString;
 }
```

AKTU PYQs

Web Technology (AKTU 2021 – 22)

Design a JavaScript program to display the current day and time in the following format.
Today is : Monday.
Current time is : 11 AM : 50 : 58

```
    // Call the function once to display the initial time
    displayDateAndTime();

    // Update the time every second
    setInterval(displayDateAndTime, 1000);
</script>
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-12

Today's Target

- JavaScript
  - Introduction to AJAX
    - Features
    - Working
    - 'XMLHttpRequest' Object
      - Methods
      - Properties
    - Example
  - AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

JavaScript

AJAX

***AJAX* stands for *Asynchronous JavaScript and XML*. It allows web pages to update without reloading the entire page.**

Key Features:

Asynchronous Communication:

Fetch data in the background while users interact with the page.

Partial Page Updates:

Only the necessary part of the web page is updated.

Reduces Server Load:

Sends and receives only required data, minimizing bandwidth usage.

JavaScript

AJAX

Working of AJAX

1. *User Action*:

   Triggers an event (e.g., clicking a button).

2. *AJAX Request*:

   JavaScript sends a request to the server.

3. *Server Response*:

   Server processes the request and sends back data.

4. *Update Page*:

   JavaScript dynamically updates the web page with the received data.

JavaScript

AJAX

'XMLHttpRequest' Object

The XMLHttpRequest object is used in JavaScript to *send and receive data from a web server asynchronously*, allowing web pages to update dynamically without reloading the entire page.

It plays a key role in implementing AJAX (Asynchronous JavaScript and XML) functionality.

'XMLHttpRequest' Object Creation

Using new keyword along with XMLHttpRequest() constructor we may create a new XMLHttpRequest object.

Example:

```
let xhr = new XMLHttpRequest()
```

JavaScript

AJAX

‘XMLHttpRequest’ Object Methods

S. No.	Method	Description	Example
1	open(method, url, async)	Initializes a request with the specified HTTP method, URL, and asynchronous flag.	xhr.open('GET', 'https://example.com/api', true);

2	send(data)	Sends the request to the server. Optionally, data can be included in POST requests.	xhr.send(); // For GET
			xhr.send(JSON.stringify({ name: 'John' })); // For POST

3	setRequestHeader(header, value)	Sets a custom HTTP header for the request.	xhr.setRequestHeader('Content-Type', 'application/json');
4	getResponseHeader(header)	Retrieves the value of a specific HTTP response header.	console.log(xhr.getResponseHeader('Content-Type'));
5	getAllResponseHeaders()	Returns all the response headers as a string.	console.log(xhr.getAllResponseHeaders());
6	abort()	Cancels the ongoing HTTP request.	xhr.abort();

JavaScript

AJAX

'XMLHttpRequest' Object Properties

S. No.	Property	Description
1	onreadystatechange	Defines a function to be called when the readyState property changes
2	readyState	Holds the status of the XMLHttpRequest. 0: Request not initialized 1: Server connection established 2: Request received 3: Processing request 4: Request finished and response is ready
3	responseText	Returns the response data as a string
4	responseXML	Returns the response data as XML data
5	status	Returns the status-number of a request 200: "OK" 404: "Not Found"
6	statusText	Returns the status-text (e.g. "OK" or "Not Found")

JavaScript

AJAX

Example:

```
<html>
  <body>
    <script>
      function displayDoc() {
        // Creating XMLHttpRequest object
        var myObj = new XMLHttpRequest();
        // Creating a callback function
        myObj.onreadystatechange = function() {
          if (this.readyState == 4 && this.status == 200) {
            document.getElementById("sample").innerHTML = this.responseText;
          }
        };
        // Open the given file
        myObj.open("GET", "http://localhost:8000/data.json", true);
        // Sending the request to the server
        myObj.send();
      }
    </script>
```

JavaScript

AJAX

Example:

```
                <h2>Getting Data</h2>
                <p>Please click on the button to fetch data</p>
                <button type="button" onclick="displayDoc()">Click Me</button>
                <div id="sample">
                </div>
        </body>
</html>
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-13

Today's Target

- Networking
  - Internet Addressing
  - InetAddress
    - Factory Methods
    - Instance Methods
  - URL
  - URLConnection
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Web Technology

UNIT – III:

Scripting:

JavaScript: Introduction, documents, forms, statements, functions, objects, Introduction to AJAX.

Networking:

Internet Addressing, InetAddress, Factory Methods, Instance Methods, TCP/IP Client Sockets, URL, URL Connection, TCP/IP Server Sockets, Datagram.

AKTU PYQs

Web Technology (AKTU 2022 – 23)

1. Discuss about Java Socket Programming. Write a program in Java to create client and server using Socket, ServerSocket class.

Web Technology (AKTU 2021 – 22)

1. Create a Java program to find out the IP address of your machine.

Web Technology (AKTU 2020 – 21)

1. Describe factory method.
2. Describe TCP/IP Client and Server Sockets with suitable diagram.

Web Technology (AKTU 2019 – 20)

1. Discuss Socket and ServerSocket in Java with its package. Write a program in Java to demonstrate how the communication is established between client and server.

Networking

"Networking refers to connecting multiple devices or systems to share information and resources."

Example: Browsing a website, where your device communicates with a server over the internet.

Networking in Java

- Java is a popular programming language for *Network or Internet programming*.
- The reason is its ability to generate *secure, cross platform, portable code*.
- Java supports network programming through the classes defined in the *java.net* package.
- They provide an *easy-to-use means* by which programmers of all skill levels can access network resources.

Networking

Internet Addressing

An *Internet address* is a number that *uniquely identifies each computer/ device* on the Internet.

- ➢ Originally, all Internet addresses consisted of 32-bit values, organized as four 8-bit values. This address type was specified by *IPv4* (Internet Protocol, version 4).
- ➢ A new addressing scheme, called IPv6 uses a 128-bit value to represent an address, organized into eight 16-bit chunks.
- ➢ IPv6 supports a much *larger address space* than does IPv4.
- ➢ When using Java, we need not to worry about whether IPv4 or IPv6 addresses are used because Java handles the underlying details for programmers.
- ➢ The name of an Internet address, called its *domain name*, describes a machine's location as a name.
- ➢ An Internet domain name is mapped to an IP address by the *Domain Name System (DNS)*.
- ➢ This enables users to work with domain names, provided Internet actually operates on IP addresses.

Networking

InetAddress

In Java, the InetAddress class (contained in java.net package) represents Internet Address objects. It encapsulates both the numerical IP address and the domain name for that address.

- You interact with this class by using the name of an IP host, which is more convenient and understandable than its IP address. The InetAddress class hides the number inside.
- InetAddress can handle both IPv4 and IPv6 addresses.

Factory Methods

- Factory Methods are used to create an InetAddress object as the InetAddress class has no visible constructors.
- Factory methods are a convention whereby static methods within a class are used to create and return instances of that class. This approach is preferred over using overloaded constructors with multiple parameter options, as distinct method names can make the purpose and behavior of each method more understandable.

Networking

InetAddress

Factory Methods

Three commonly used InetAddress factory methods are shown here:

1. static InetAddress ***getLocalHost*** ( ) throws UnknownHostException

    returns the InetAddress object that represents the local host.

2. static InetAddress ***getByName*** (String hostName) throws UnknownHostException

    returns an InetAddress for a host name passed to it.

3. static InetAddress[ ] ***getAllByName*** (String hostName) throws UnknownHostException

    returns an array of InetAddresses that represent all of the addresses that a particular name resolves to. On the Internet, it is common for a single name to be used to represent several machines.

Note: If these methods are unable to resolve the host name, they throw an UnknownHostException.

Networking

InetAddress

Factory Methods

Example:

```
import java.net.InetAddress;
import java.net.UnknownHostException;

public class InetAddressFactoryMethodsDemo {

        public static void main(String[] args) throws UnknownHostException {
                InetAddress address = InetAddress.getLocalHost();
                System.out.println(address);

                address = InetAddress.getByName("courses.gatewayclasses.in");
                System.out.println(address);

                InetAddress[] addresses = InetAddress.getAllByName("courses.gatewayclasses.in");
                for (InetAddress add: addresses) {
                        System.out.println(add);
                }
        }
}
```

Networking

InetAddress

Instance Methods

The InetAddress class has several other methods, which can be *used on the objects returned by the factory methods*:

boolean ***equals***(Object other)	Returns true if this object has the same Internet address as other.
byte[ ] ***getAddress***( )	Returns a byte array that represents the object's IP address in network byte order.
String ***getHostAddress***( )	Returns a string that represents the host address associated with the InetAddress object.
String ***getHostName***( )	Returns a string that represents the host name associated with the InetAddress object.
String ***toString***( )	Returns a string that lists the host name and the IP address for convenience.

Networking

InetAddress

Instance Methods

Example:

```
import java.net.*;
public class InetAddressInstanceMethodsDemo {
        public static void main(String[] args) throws UnknownHostException {
                InetAddress localHost = InetAddress.getLocalHost();
                InetAddress address = InetAddress.getByName("courses.gatewayclasses.in");

                byte[] addressBytes = address.getAddress();
                for(byte addressByte: addressBytes)
                        System.out.println(addressByte);

                System.out.println(address.getHostAddress());
                System.out.println(address.getHostName());

                boolean isEqual = address.equals(localHost);
                System.out.println(isEqual);

                System.out.println(localHost.toString());
                System.out.println(address);
        }
}
```

Networking

Uniform Resource Locator (URL)

The URL is a reference (an address) to a resource on the Internet. Every browser uses them to identify information on the Web.

A URL has two main components

1. Protocol Identifier
2. Resource Name

Example : http://java.sun.com/

http	: Protocol Identifier
java.sun.com/	: Resource Name

The Resource Name may contain

Host Name	: The name of the machine
File Name	: The pathname of the file on the machine
Port Number	: The port number to which to connect (Optional)
Reference	: A reference to a named anchor within a resource (Optional)

Networking

'URL' class in Java

The URL class in Java represents a Uniform Resource Locator, which is an address of a resource on the web and provides methods to access and manipulate its components like protocol, host, port, and file path.

Constructors

Java's URL class has several constructors:

- One commonly used form specifies the URL with a string same as the one used in a browser:

  URL(String urlSpecifier) throws MalformedURLException

- The other two constructors allow us to break up the URL into its component parts:

  URL(String protocolName, String hostName, int port, String path) throws MalformedURLException

  URL(String protocolName, String hostName, String path) throws MalformedURLException

Networking

'URL' class in Java

Instance Methods

The following methods, called with an instance of URL class, parse a URL string into its components (protocol, host, port, file, etc.).

String *getProtocol*():	Returns the protocol of the URL (e.g., http, https, ftp).
String *getHost*():	Returns the hostname or IP address of the URL.
int *getPort*():	Returns the port number specified in the URL, or -1 if none is explicitly mentioned.
String *getFile*():	Returns the file name or path specified in the URL, including the query string if present.
String *getRef*():	Returns the reference (anchor) part of the URL, typically after the # symbol.

Networking

URL

Instance Methods

Parsing a URL - Example:

```
import java.net.*;
public class URLDemo {
        public static void main(String[]args) throws MalformedURLException {
                URL url = new
        URL("http://java.sun.com:80/docs/books/tutorial/intro.html#DOWNLOADING");
                System.out.println("Protocol = "+url.getProtocol());
                System.out.println("Host     = "+url.getHost());
                System.out.println("Port     = "+url.getPort());
                System.out.println("FileName = "+url.getFile());
                System.out.println("Reference= "+url.getRef());
        }
}
```

Networking

'URLConnection' class in Java

The `URLConnection` class in Java represents a communication link between an application and a URL, allowing interaction with the resource referenced by the URL. It provides methods for reading from and writing to the resource, managing headers, and configuring the connection.

Once you make a connection to a remote server, you can use URLConnection to inspect the properties of the remote object before actually transporting it locally.

URLConnection defines several methods:

int getContentLength( )	Returns the size in bytes of the content associated with the resource.
String getContentType( )	Returns the type of content found in the resource. This is the value of the content-type header field.
long getDate( )	Returns the time and date of the response.
long getExpiration( )	Returns the expiration time and date of the resource.

Networking

‘URLConnection’ class in Java

URLConnection defines several methods:

String getHeaderField(int idx)	Returns the value of the header field at index idx.
String getHeaderField(String fieldName)	Returns the value of header field whose name is specified by fieldName.
String getHeaderFieldKey(int idx)	Returns the header field key at index idx.
Map<String, List<String>> getHeaderFields( )	Returns a map that contains all of the header fields and values.
long getLastModified( )	Returns the time and date of the last modification of the resource.
InputStream getInputStream( ) throws IOException	Returns an InputStream that is linked to the resource. This stream can be used to obtain the content of the resource.

Networking

'HttpURLConnection' class in Java

HttpURLConnection a subclass of URLConnection that provides support for HTTP connections. You obtain an HttpURLConnection by calling openConnection( ) on a URL object, but you must cast the result to HttpURLConnection.

HttpURLConnection defines several methods:

String getRequestMethod( )	Returns a string representing how URL requests are made. The default is GET. Other options, such as POST, are also available.
int getResponseCode( ) throws IOException	Returns the HTTP response code
String getResponseMessage( ) throws IOException	Returns the response message associated with the response code.
void setRequestMethod(String how) throws ProtocolException	Sets the method by which HTTP requests are made to that specified by how.

Networking

URLConnection & HttpURLConnection

Example:

```
import java.io.*;
import java.net.*;
import java.util.Date;

public class URLConnectionDemo {
        public static void main(String[] args) throws IOException {
                int c;
                URL url = new URL("http://google.co.in/");
                HttpURLConnection urlConnection = (HttpURLConnection)
                                                        url.openConnection();
                System.out.println(urlConnection.getContentLength());
                System.out.println(urlConnection.getContentType());
                System.out.println(new Date(urlConnection.getDate()));
                System.out.println(new Date(urlConnection.getExpiration()));
                System.out.println(new Date(urlConnection.getLastModified()));
```

Networking

URLConnection & HttpURLConnection

Example:

```
                    System.out.println(urlConnection.getContentLengthLong());
                    System.out.println(urlConnection.getResponseCode());
                    System.out.println(urlConnection.getResponseMessage());
                    System.out.println(urlConnection.getRequestMethod());
                    System.out.println();

                    InputStream input = urlConnection.getInputStream();
                    while (((c = input.read()) != -1)) {
                    System.out.print((char) c);
                    }
                    input.close();
            }

    }
```

AKTU

B.Tech 5th Sem

GATEWAY CLASSES

CS IT & CS Allied

Web Technology

UNIT-3 Lecture-14

Today's Target

- Networking
  - TCP/IP Client Sockets
  - TCP/IP Server Sockets
  - Datagram
- AKTU PYQs

By Amol Sharma sir

- Pursuing Ph.D. from IIT (BHU)
- Wipro Certified Faculty (WCF)

Web Technology

UNIT – III:

Scripting:

JavaScript: Introduction, documents, forms, statements, functions, objects, Introduction to AJAX.

Networking:

Internet Addressing, InetAddress, Factory Methods, Instance Methods, TCP/IP Client Sockets, URL, URL Connection, TCP/IP Server Sockets, Datagram.

Networking

Socket

"A socket is an endpoint for communication between two machines over a network."

A socket is essentially the combination of:

IP Address: Identifies the specific device on the network.
Port Number: Identifies a specific process or service running on that device.

Together, these ensure that data is delivered to the correct application on the correct machine. This combination is often referred to as a *socket address*.

Example: ***192.168.1.10:8080***

This socket refers to the application listening on port 8080 of the machine with IP address 192.168.1.10.

Networking

Socket

Types of Sockets:

Stream Sockets (TCP Sockets)

Use TCP (Transmission Control Protocol) for reliable, connection-oriented communication.

Example: In Java, the Socket class represents a client-side TCP Socket, while the ServerSocket class represents a server socket.

Datagram Sockets (UDP Sockets)

Use UDP (User Datagram Protocol) for connectionless, faster but less reliable communication.

Example: In Java, the DatagramSocket class is used for sending and receiving packets over a network using UDP (User Datagram Protocol)

These sockets allow communication tailored to different network needs—reliability (TCP) or speed (UDP).

Networking

Socket Programming

"Socket Programming is a technique in network programming that allows two devices (processes) to communicate over a network."

It involves *creating endpoints, called sockets*, on both client and server sides to establish a connection for exchanging data.

In Java, Socket Programming enables the implementation of both TCP (reliable, connection-oriented) and UDP (fast, connectionless) communication using classes like Socket, ServerSocket, and DatagramSocket.

Networking

Socket Programming with TCP

Sockets for server and client

Server

Welcoming socket welcomes some initial contact from a client.
Connection socket is created at initial contact of client. This new socket is dedicated to the particular client.

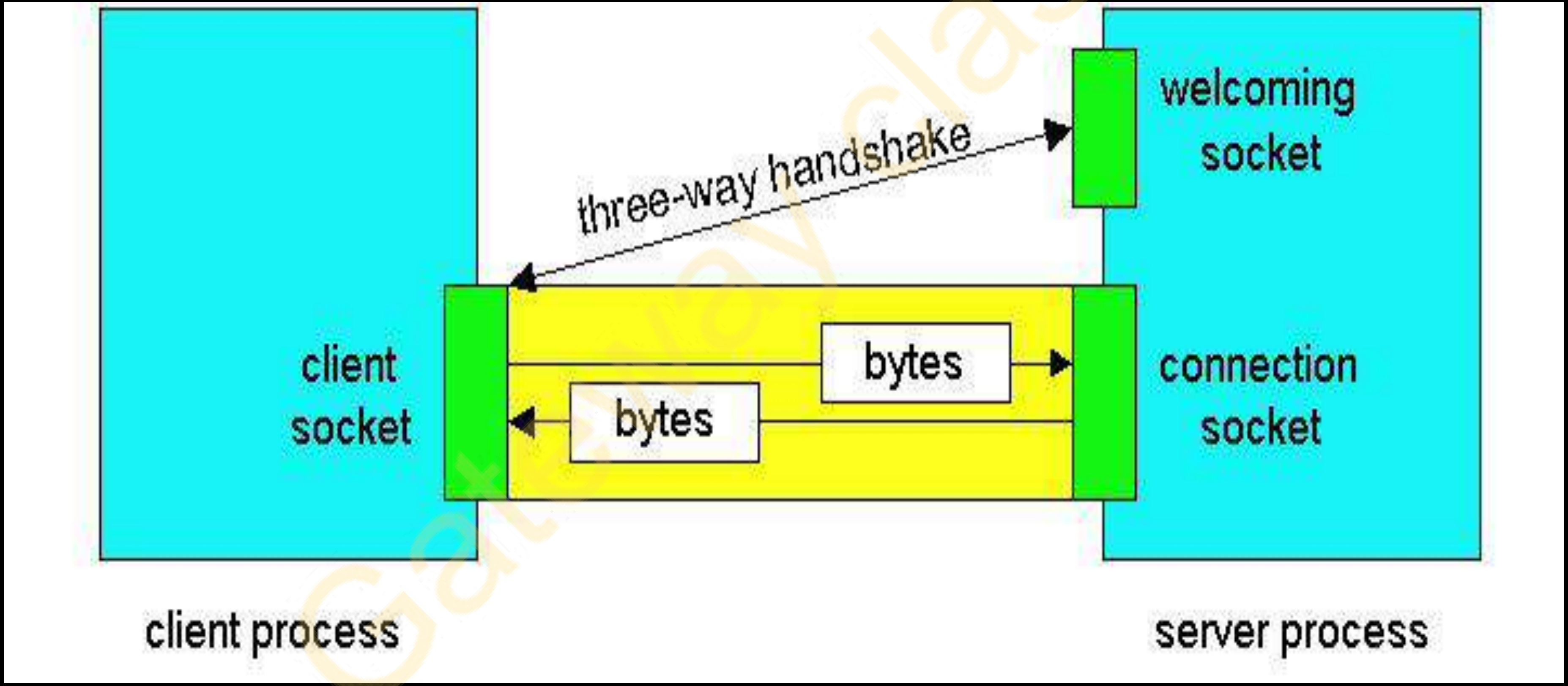
Client

Client socket initiates a TCP connection to the server by creating a socket object. It specifies the address of the server process, namely, the IP address of the server and the port number of the process.

Networking

Socket Programming with TCP

Sockets for server and client



Networking

Socket Programming with TCP

Java TCP Sockets

In Package java.net

Socket

Implements client sockets (also called just "sockets").
An endpoint for communication between two machines.

Constructor and Methods

Socket(String host, int port): Creates a stream socket and connects it to the specified port number on the named host.
InputStream getInputStream() Returns an input stream to receive data from the connected socket.
OutputStream getOutputStream() Returns an output stream to send data through the connected socket.
close() Closes the socket and releases all associated resources.

Networking

Socket Programming with TCP

Java TCP Sockets

In Package java.net

ServerSocket

Implements server sockets.

Waits for requests to come in over the network. Performs some operation based on the request.

Constructor and Methods

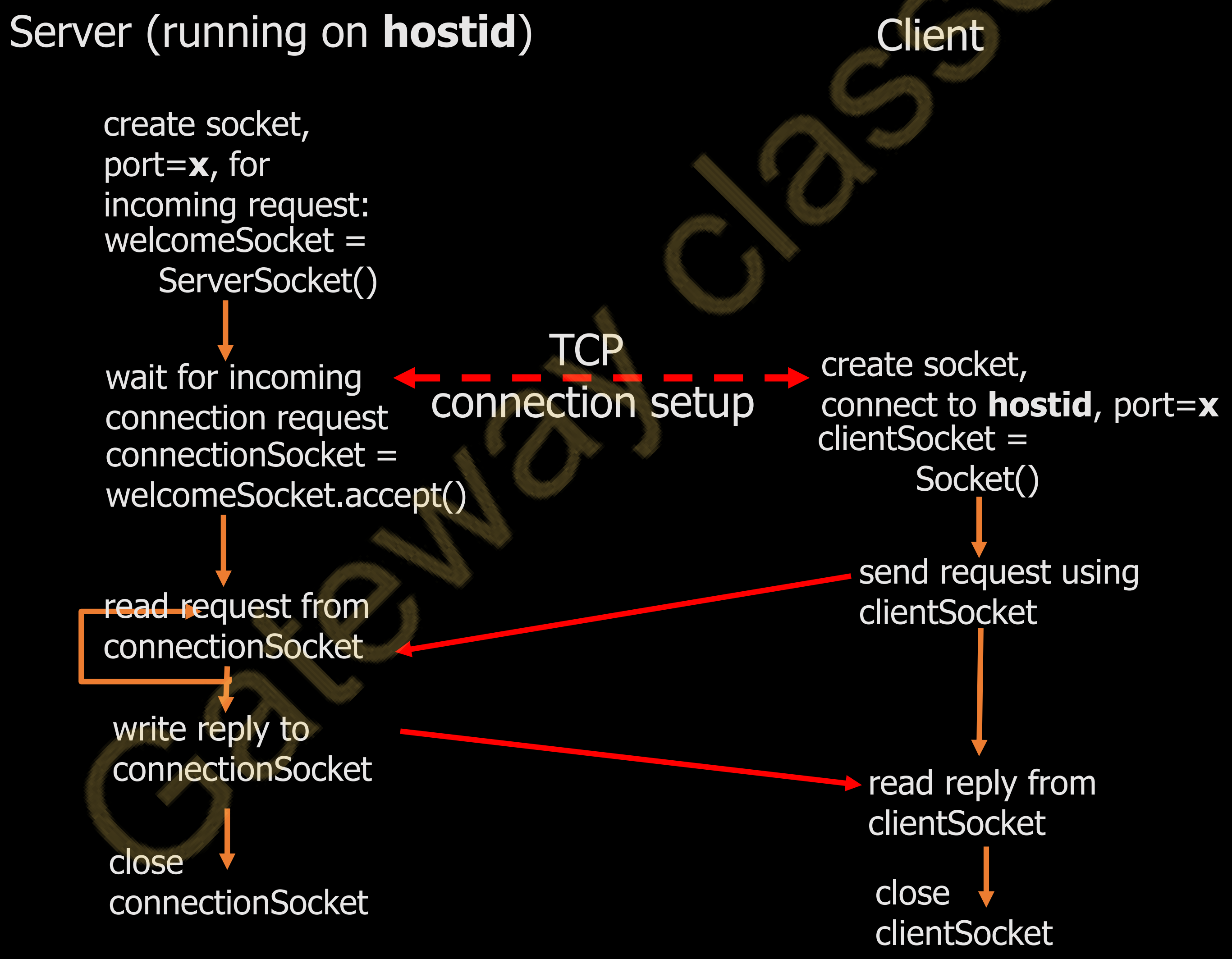
ServerSocket(int port): Creates a server socket on the specified port.

Socket accept(): listens for incoming client connections and returns a Socket object when a client successfully connects.

Networking

Socket Programming with TCP

TCP Client/server socket interaction



Server (running on **hostid**)
Client
create socket, port=**x**, for incoming request: welcomeSocket = ServerSocket()
wait for incoming connection request connectionSocket = welcomeSocket.accept()
TCP connection setup
create socket, connect to **hostid**, port=**x** clientSocket = Socket()
send request using clientSocket
read request from connectionSocket
write reply to connectionSocket
read reply from clientSocket
close connectionSocket
close clientSocket

Networking

Socket Programming with TCP

Java TCP Sockets

Example:

TCPClient.java

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("localhost", 6789);
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new
                                 InputStreamReader(clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}
```

Networking

Socket Programming with TCP

Java TCP Sockets

Example: *TCPServer.java*

```
import java.io.*;
import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);
        while (true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(new
                      InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new
                      DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            System.out.println("Request Data: " + clientSentence);
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            System.out.println("Response Data: " + capitalizedSentence);
            outToClient.writeBytes(capitalizedSentence);
        }
    }}
```

Networking

Datagrams

"*Datagrams* are bundles of information passed between machines."

- ➢ Once the datagram has been released to its intended target, there is no assurance that it will arrive or even that someone will be there to receive it.
- ➢ Likewise, when the datagram is received, there is no assurance that it hasn't been damaged in transit or that whoever sent it is still there to receive a response.

Java implements datagrams on top of the UDP protocol by using two classes:

1. The *DatagramPacket* object is the data container, while
2. The *DatagramSocket* is the mechanism used to send or receive the DatagramPacket.

Networking

User Datagram Protocol (UDP) can be used directly to support fast, connectionless, unreliable transport of packets.

- Connectionless and unreliable service.
- There is no initial handshaking phase.
- The transmitted data may be received out of order, or lost

Socket Programming with UDP

- No need for a welcoming socket.
- No streams are attached to the sockets.
- The sending hosts creates "packets" by attaching the IP destination address and port number to each batch of bytes.
- The receiving process must untie the received packet to obtain the packet's information bytes.

Networking

Socket Programming with UDP

Java UDP Sockets (Datagram Sockets)

In Package java.net

DatagramSocket

A socket for sending and receiving datagram packets.

Constructor and Methods

DatagramSocket(int port):

Constructs a datagram socket and binds it to the specified port on the local host machine.

void receive(DatagramPacket p)

waits for a packet to be received and returns the result.

void send(DatagramPacket p)

sends a packet to the port specified by packet

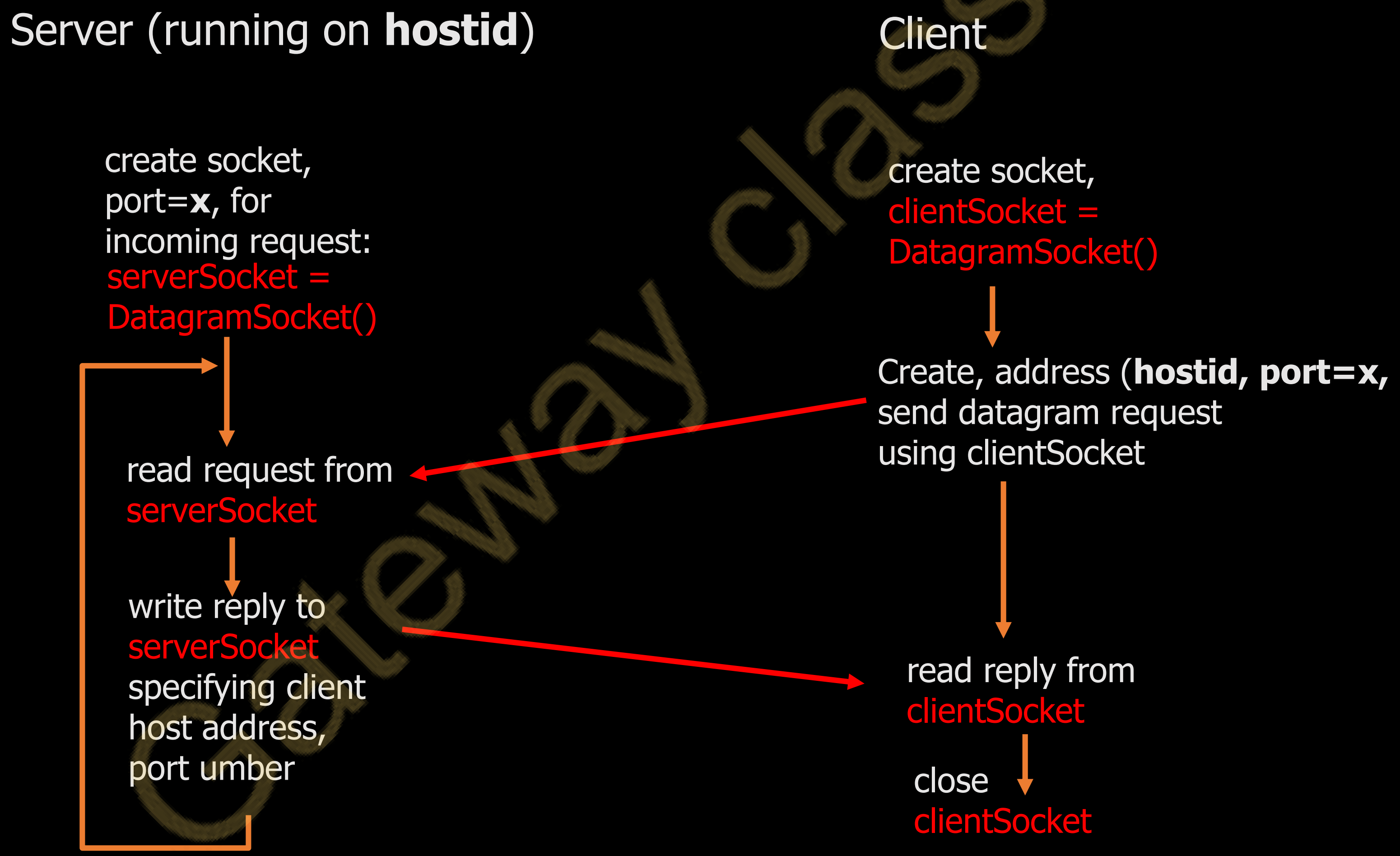
void close()

closes the socket.

Networking

Socket Programming with TCP

UDP Client/server socket interaction



Server (running on **hostid**)

create socket,
port=**x**, for
incoming request:
serverSocket =
DatagramSocket()

read request from
serverSocket

write reply to
serverSocket
specifying client
host address,
port umber

Client

create socket,
clientSocket =
DatagramSocket()

Create, address (**hostid, port=x,**
send datagram request
using clientSocket

read reply from
clientSocket

close
clientSocket

Networking

Socket Programming with UDP

Java Datagram Sockets

Example:

UDPClient.java

```
import java.io.*;
import java.net.*;
class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser =new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("localhost");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket =  new DatagramPacket(sendData,sendData.length,IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
         String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence.trim());
        clientSocket.close();
    }
}
```

Networking

Socket Programming with UDP

Java Datagram Sockets

Example: *UDPServer.java*

```
import java.io.*; import java.net.*;
class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];         byte[] sendData = new byte[1024];
        String endString="   ";
        while(!(endString.substring(0,3).equalsIgnoreCase("Bye"))) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData,receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData());
            System.out.println("FROM CLIENT: "+sentence.trim());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            endString=sentence;
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                                                                    IPAddress, port);
            serverSocket.send(sendPacket);
        } serverSocket.close();
    } }
```

AKTU PYQs

Web Technology (AKTU 2022 – 23)

1. Discuss about Java Socket Programming. Write a program in Java to create client and server using Socket, ServerSocket class.

Web Technology (AKTU 2021 – 22)

1. Create a Java program to find out the IP address of your machine.

Web Technology (AKTU 2020 – 21)

1. Describe factory method.
2. Describe TCP/IP Client and Server Sockets with suitable diagram.

Web Technology (AKTU 2019 – 20)

1. Discuss Socket and ServerSocket in Java with its package. Write a program in Java to demonstrate how the communication is established between client and server.

Thank you