Basics of JavaScript Programming

Unit - I

Difference between Scripting & Programming Languages

- Basically all the scripting languages are the programming languages.
- The theoretical difference between the two is that scripting languages do not require the compilation step and are rather interpreted.
- Generally, compiled programs run faster than interpreted programs because they are first converted native machine code.

Difference between Scripting & Programming Languages

- Also, compilers read and analyze the code only once, and report the errors collectively that the code might have, but the interpreter will read and analyze the code statements each time it meets them and halts at that very instance if there is some error.

- Some scripting languages traditionally used without an explicit compilation step are JavaScript, PHP, Python, VBScript.

- Some programming languages traditionally used with an explicit compilation step are C, C++.

Difference between Scripting & Programming Languages

- **Applications of Scripting Languages :**
  - To automate certain tasks in a program
  - Extracting information from a data set
  - Less code intensive as compared to traditional programming languages

- **Applications of Programming Languages :**
  - They typically run inside a parent program like scripts
  - More compatible while integrating code with mathematical models
  - Languages like JAVA can be compiled and then used on any platform

Client Side Vs Server Side Scripting

- Website scripts run in one of two places – the client side, also called the front-end, or the server side, also called the back-end.

- The client of a website refers to the web browser that is viewing it. The server of a website is, of course, the server that hosts it.

Client Side Vs Server Side Scripting

- Client side script is the program that runs on the client machine (browser) and deals with the user interface/display and any other processing that can happen on client machine like reading/writing cookies.

- The Programming languages for client-side programming are :
  1) Javascript
  2) VBScript
  3) HTML
  4) CSS
  5) AJAX

Client Side Vs Server Side Scripting

- Server side script is the program that runs on server dealing with the generation of content of web page.

- The Programming languages for server-side programming are :
  1) PHP
  2) C++
  3) Java and JSP
  4) Python
  5) Ruby on Rails

Java Script

- Java script is a scripting language.
- Scripting language is small snippets that can run as a part of a bigger HTML document.
- This language is designed primarily for building web pages. Java script can be used along with HTML to create better web pages.

JavaScript

- Javascript programs are embedded within HTML documents in source code form.
- Javascript code added to HTML can perform a wide variety of functions:
  - Decision making
  - Submitting forms
  - Performing complex mathematical calculations or string manipulation.
  - Data entry validation etc.

JavaScript

- Java script is Object Oriented. It allows interaction with the properties of objects that it recognizes.
  - Internal built-in objects (e.g. Window object: this object can be used to manipulate the way current output window/frame in the browser displays its content)
  - Browser Object (e.g. Document object can be used to manipulate a document properties like color, font, size etc)

JavaScript

- Java script can run on both client side (by the browser after downloading the page) and server side (with the help of Livewire environment)

Difference between Java & JavaScript

- Although the name resemble quite closely, they are different languages:
  - Both are object oriented languages.
  - Javascript programs are interpreted in source code form.
  - Java programs are first compiled into a device independent byte code, which is then interpreted.
  - Javascript is a small language and doesn't have many features that exist in java.
  - Java is a powerful language and can be used in extremely sophisticated applications.

Literals in Javascript

- A *literal is a data value that appears directly in a program.*

```
12                  // The number twelve
1.2                 // The number one point two
"hello world"       // A string of text
'Hi'                // Another string
true                // A Boolean value
false               // The other Boolean value
/javascript/gi      // A "regular expression" literal (for pattern matching)
null                // Absence of an object
```

Identifier in Javascript

- An *identifier is simply a name. In JavaScript, identifiers are used to name variables and* functions and to provide labels for certain loops in JavaScript code.

- A JavaScript identifier must begin with a letter, an underscore (_), or a dollar sign ($). Subsequent characters can be letters, digits, underscores, or dollar signs.

- (Digits are not allowed as the first character so that JavaScript can easily distinguish identifiers from numbers.)

Identifier in Javascript

- E.g.
  - i
  - my-_variable_name
  - v13
  - _dummy
  - $str

Reserved words in Javascript

- JavaScript reserves a number of identifiers as the keywords of the language itself.

- You cannot use these words as identifiers in your programs:

break	delete	function	return	typeof
case	do	if	switch	var
catch	else	in	this	void
continue	false	instanceof	throw	while
debugger	finally	new	true	with
default	for	null	try	

Variables in Javascript

- A variable should be declared before it is used in Javascript.

- Variables are declared using the keyword *var*
  e.g. var i;

- Variable initialization and declaration can be combined like this:
  e.g. var i=0;

- If a variable is declared but not initialized then its initial value is undefined.

Variables in Javascript

- The *scope of a variable is the region of your program source code in which it is defined.*

- A *global variable has global scope; it is defined everywhere in your JavaScript code.*

- *On* the other hand, variables declared within a function are defined only within the body of the function. They are *local variables and have local scope.*

Variables in Javascript

- *Function parameters also* count as local variables and are defined only within the body of the function.

- Within the body of a function, a local variable takes precedence over a global variable with the same name.

Operators in Javascript

Operator	Operation	A	N	Types
++	Pre- or post-increment	R	1	lval→num
--	Pre- or post-decrement	R	1	lval→num
-	Negate number	R	1	num→num
+	Convert to number	R	1	num→num
~	Invert bits	R	1	int→int
!	Invert boolean value	R	1	bool→bool

Operators in Javascript

Operator	Operation	A	N	Types
delete	Remove a property	R	1	lval→bool
typeof	Determine type of operand	R	1	any→str
void	Return undefined value	R	1	any→undef
*, /, %	Multiply, divide, remainder	L	2	num,num→num
+, -	Add, subtract	L	2	num,num→num
+	Concatenate strings	L	2	str,str→str
<<	Shift left	L	2	int,int→int
>>	Shift right with sign extension	L	2	int,int→int
>>>	Shift right with zero extension	L	2	int,int→int
<, <=,>, >=	Compare in numeric order	L	2	num,num→bool
<, <=,>, >=	Compare in alphabetic order	L	2	str,str→bool
instanceof	Test object class	L	2	obj,func→bool
in	Test whether property exists	L	2	str,obj→bool
==	Test for equality	L	2	any,any→bool
!=	Test for inequality	L	2	any,any→bool
===	Test for strict equality	L	2	any,any→bool
!==	Test for strict inequality	L	2	any,any→bool
&	Compute bitwise AND	L	2	int,int→int
^	Compute bitwise XOR	L	2	int,int→int
|	Compute bitwise OR	L	2	int,int→int

Operators in Javascript

&&	Compute logical AND	L	2	any,any→any
||	Compute logical OR	L	2	any,any→any
? :	Choose 2nd or 3rd operand	R	3	bool,any,any→any
=	Assign to a variable or property	R	2	lval,any→any
*=, /=, %=, +=, -=, &=, ^=, |=, <<=, >>=, >>>=	Operate and assign	R	2	lval,any→any
,	Discard 1st operand, return second	L	2	any,any→any

Expressions in Javascript

- **Primary Expressions:**
- The simplest expressions, known as *primary expressions, are those that stand alone*—they do not include any simpler expressions.

- Primary expressions in JavaScript are *constant* or *literal values*, *certain language keywords, and variable references.*

E.g.

```
1.23          // A number literal
"hello"       // A string literal
/pattern/     // A regular expression literal
```

Expressions in Javascript

- **Primary Expressions:**
- Some of JavaScript's *reserved words* are primary expressions:

  E.g.

```
true        // Evalutes to the boolean true value
false       // Evaluates to the boolean false value
null        // Evaluates to the null value
this        // Evaluates to the "current" object
```

Expressions in Javascript

- **Primary Expressions:**
- Finally, the third type of primary expression is the *bare variable* reference:

```
i           // Evaluates to the value of the variable i.
sum         // Evaluates to the value of the variable sum.
undefined   // undefined is a global variable, not a keyword like null.
```

Expressions in Javascript

- **Object and Array Initializers:**

- Object and array *initializers are expressions whose value is a newly created object or* array.

- These initializer expressions are sometimes called "*object literals*" and "*array literals*."

- Unlike true literals, however, they are not primary expressions, because they include a number of sub expressions that specify property and element values.

Expressions in Javascript

- **Object and Array Initializers:**

- An *array initializer* is a comma-separated list of expressions contained within square brackets.

- The value of an array initializer is a newly created array. The elements of this new array are initialized to the values of the comma-separated expressions:

```
[]              // An empty array: no expressions inside brackets means no elements
[1+2,3+4]   // A 2-element array. First element is 3, second is 7
var matrix = [[1,2,3], [4,5,6], [7,8,9]];
```

Expressions in Javascript

- **Object and Array Initializers:**

- Object initializer expressions are like array initializer expressions, but the square brackets are replaced by curly brackets, and each subexpression is prefixed with a property name and a colon:
  e.g.

```
var p = { x:2.3, y:-1.2 };     // An object with 2 properties
var q = {};                    // An empty object with no properties
q.x = 2.3; q.y = -1.2;         // Now q has the same properties as p
```

Expressions in Javascript

- **Function Definition Expressions:**
- A function definition expression defines a JavaScript function, and such an expression is the newly defined function.

- A function definition expression typically consists of the keyword function followed by a comma-separated list of zero or more identifiers (the parameter names) in parentheses and a block of JavaScript code (the function body) in curly braces. For example:

```
// This function returns the square of the value passed to it.
var square = function(x) { return x * x; }
```

A function definition expression can also include a name for the function.

Expressions in Javascript

- **Property Express Expressions:**
  - A property access expression evaluates to the value of an obje property or an array element.

  - JavaScript defines two syntaxes for property access:

    - *expression . identifier*
    - *expression [expression]*

Expressions in Javascript

- **Property Express Expressions:**
  - *expression . Identifier*

  - The first style of property access is an expression followed by a period and an identifier.

  - The expression specifies the object, and the identifier specifies the name of the desired property.

E.g.

```
var o = {x:1,y:{z:3}};        // An example object
var a = [o,4,[5,6]];          // An example array that contains the object
o.x                           // => 1: property x of expression o
```

Expressions in Javascript

- **Property Express Expressions:**
  - *expression [expression]*

- The second style of property access follows the first expression (the object or array) with another expression in square brackets.

- This second expression specifies the name of the desired property of the index of the desired array element.

Expressions in Javascript

- **Property Express Expressions:**
- *expression [expression]*

E.g.

```
var o = {x:1,y:{z:3}};          // An example object
var a = [o,4,[5,6]];            // An example array that contains the object

o.y.z                           // => 3: property z of expression o.y
o["x"]                          // => 1: property x of object o
a[1]                            // => 4: element at index 1 of expression a
a[2]["1"]                       // => 6: element at index 1 of expression a[2]
a[0].x                          // => 1: property x of expression a[0]
```

Expressions in Javascript

- **Property Express Expressions:**
- The *.identifier syntax is the simpler of the two property access options, but notice* that it can only be used when the property you want to access has a name that is a legal identifier, and when you know then name when you write the program.

- If the property name is a reserved word or includes spaces or punctuation characters, or when it is a number (for arrays), you must use the square bracket notation.

Expressions in Javascript

- **Invocation Expressions:**
  - An *invocation expression is JavaScript's syntax for calling (or executing) a function or* method. It starts with a function expression that identifies the function to be called.

  - The function expression is followed by an open parenthesis, a comma-separated list of zero or more argument expressions, and a close parenthesis.

e.g.

```
f(0)              // f is the function expression; 0 is the argument expression.
Math.max(x,y,z) // Math.max is the function; x, y and z are the arguments.
a.sort()          // a.sort is the function; there are no arguments.
```

Expressions in Javascript

- **Invocation Expressions:**
  - Every invocation expression includes a pair of parentheses and an expression before the open parenthesis.
  - If that expression is a property access expression, then the invocation is known as a *method invocation.*
  - In method invocations, the object or array that is the subject of the property access becomes the value of the this parameter while the body of the function is being executed.
  - This enables an object-oriented programming paradigm in which functions (known by their OO name, “methods”) operate on the object of which they are part.

Control Statements in Javascript

- **If statement:**
  - The if statement is the fundamental control statement that allows JavaScript to make decisions, or, more precisely, to execute statements conditionally.

  - This statement has two forms. The first is:

    if (expression)

    statement
  - In this form, *expression is evaluated. If the resulting value is true, statement is executed.*
  - If *expression is false, statement is not executed.*

Control Statements in Javascript

- **If statement:**
- E.g.

```
if (username == null)            // If username is null or undefined,
username = "John Doe";           // define it
```

Or similarly:

```
// If username is null, undefined, false, 0, "", or NaN, give it a new value

if (!username) username = "John Doe";
```

Control Statements in Javascript

- **If else statement:**
  - The second form of the if statement introduces an else clause that is executed when *expression is false. Its syntax is:*

```
if (expression)
  statement1
else
  Statement2
```

  - This form of the statement executes *statement1 if expression is trute and executes* statement2 if *expression is false.*

Control Statements in Javascript

- **If else statement:**
  - **E.g.**

```
if (n == 1)
        console.log("You have 1 new message.");
else
        console.log("You have " + n + " new messages.");
```

Control Statements in Javascript

- **Nested If else statement:**
  - If else statements within another if else statement.

```
If(expression)
{
  if(expression)
    statement 1;
  else
    statement 2;
}
Else
  statement 3
```

Control Statements in Javascript

- **else if statement:**
  - The if/else statement evaluates an expression and executes one of two pieces of code, depending on the outcome.
  - But what about when you need to execute one of many pieces of code? One way to do this is with an else if statement.

Control Statements in Javascript

- **else if statement:**

```
if (n == 1) {
        // Execute code block #1
}
else if (n == 2) {
        // Execute code block #2
}
else if (n == 3) {
        // Execute code block #3
}
else {
        // If all else fails, execute block #4
}
```

Control Statements in Javascript

- **switch statement:**
  - An if statement causes a branch in the flow of a program's execution, and you can use the else if idiom to perform a multiway branch.
  - This is not the best solution, however, when all of the branches depend on the value of the same expression.
  - In this case, it is wasteful to repeatedly evaluate that expression in multiple if statements.
  - The switch keyword is followed by an expression in parentheses and a block of code in curly braces:

Control Statements in Javascript

- **switch statement:**

```
switch(expression/value) {
        case value1: statements;
                     break;
        case value2: statements;
                     break;
        case value3: statements;
                     break;
        default:      statements;
                     break;
}
```

Loop Statements in Javascript

- **while Loop:**
  - Just as the if statement is JavaScript's basic conditional, the while statement is Java-Script's basic loop.
    It has the following syntax:

    while (expression)
      statement
  - To execute a while statement, the interpreter first evaluates *expression. If the value of* the expression is false, then the interpreter skips over the *statement that serves as the* loop body and moves on to the next statement in the program.
  - If, on the other hand, the *expression is true, the interpreter executes the statement and repeats, jumping* back to the top of the loop and evaluating *expression again.*

Loop Statements in Javascript

- **do while Loop:**
  - The do/while loop is like a while loop, except that the loop expression is tested at the bottom of the loop rather than at the top. This means that the body of the loop is always executed at least once. The syntax is:

```
do
        statement
while (expression);
```

  - There are a couple of syntactic differences between the do/while loop and the ordinary while loop.

Loop Statements in Javascript

- **do while Loop:**
  - First, the do loop requires both the do keyword (to mark the beginning of the loop) and the while keyword (to mark the end and introduce the loop condition).
  - Also, the do loop must always be terminated with a semicolon. The while loop doesn't need a semicolon if the loop body is enclosed in curly braces.

Loop Statements in Javascript

- **for Loop:**
  - The for statement provides a looping construct that is often m
    than the while statement.
  - The for statement simplifies loops that follow a common pattern.
  - Most loops have a counter variable of some kind. This variable is initialized before the loop starts and is tested before each iteration of the loop. Finally, the counter variable is incremented or otherwise updated at the end of the loop body, just before the variable is tested again.
  - In this kind of loop, the initialization, the test, and the update are the three crucial manipulations of a loop variable.

Loop Statements in Javascript

- **for Loop:**
  - The for statement encodes each of these three manipulations as an expression and makes those expressions an explicit part of the loop syntax:

for(initialize ; test ; increment)

statement

Loop Statements in Javascript

- **for in Loop:**
  - The for/in statement uses the for keyword, but it is a completely different kind of loop than the regular for loop.
  - A for/in loop looks like this:

for (variable in object)

statement

  - *variable typically names a variable, but it may be any expression that evaluates to an* lvalue or a var statement that declares a single variable—it must be something suitable as the left side of an assignment expression.

Loop Statements in Javascript

- **for in Loop:**
  - *object is an expression that evaluates* to an object.
  - As usual, *statement is the statement or statement block that serves* as the body of the loop.
  - It is easy to use a regular for loop to iterate through the elements of an array:
  - The for/in loop makes it easy to do the same for the properties of an object:

```
for(var p in o) // Assign property names of o to variable p
    console.log(o[p]); // Print the value of each property
```

Loop Statements in Javascript

- **for in Loop:**
  - To execute a for/in statement, the JavaScript interpreter first evaluates the *object expression.*
  - If it evaluates to null or undefined, the interpreter skips the loop and moves on to the next statement.
  - If the expression evaluates to a primitive value, that value is converted to its equivalent wrapper object.
  - Before each iteration, however, the interpreter evaluates the *variable expression and assigns the name of the property (a string value) to it.*

Loop Statements in Javascript

- **for in Loop:**
  - To execute a for/in statement, the JavaScript interpreter first evaluates the *object expression.*
  - If it evaluates to null or undefined, the interpreter skips the loop and moves on to the next statement.
  - If the expression evaluates to a primitive value, that value is converted to its equivalent wrapper object.
  - Before each iteration, however, the interpreter evaluates the *variable expression and assigns the name of the property (a string value) to it.*

Loop Statements in Javascript

- **for in Loop:**
  - **E.g.**

```
var o = {x:1, y:2, z:3};
var a = [], i = 0;
for(a[i++] in o) /* empty */;
```

Loop Statements in Javascript

- **for in Loop:**

```
<!DOCTYPE html>
<html>
   <body>
         <h2>JavaScript For/In Loop</h2>
         <p>The for/in statement loops through the properties of an object.</p>
         <p id="demo"></p>
         <script>
               var txt = "";
               var person = {fname:"John", lname:"Doe", age:25};
               var x;
               for (x in person) {
                 txt += person[x] + " ";
               }
               document.getElementById("demo").innerHTML = txt;
         </script>
   </body>
</html>
```

Break Statements in Javascript

- The break statement, used alone, causes the innermost enclosing loop or switch statement to exit immediately. Its syntax is simple:

```
break;
```

- Because it causes a loop or switch to exit, this form of the break statement is legal only if it appears inside one of these statements.

Break Statements in Javascript

- JavaScript also allows the break keyword to be followed by a statement label (just the identifier, with no colon):

break label_name;

- When break is used with a label, it jumps to the end of, or terminates, the enclosing statement that has the specified label.

- It is a syntax error to use break in this form if there is no enclosing statement with the specified label.

Continue Statements in Javascript

- The continue statement is similar to the break statement. Instead of exiting a loop, however, continue restarts a loop at the next iteration.

- The continue statement's syntax is just as simple as the break statement's:

  continue;

- The continue statement can also be used with a label:

  continue label_name;

Continue Statements in Javascript

- The continue statement, in both its labeled and unlabeled can be used only within the body of a loop. Using it anywhere else causes a syntax error.
- When the continue statement is executed, the current iteration of the enclosing loop is terminated, and the next iteration begins. This means different things for different types of loops:
  - In a while loop, the specified *expression at the beginning of the loop is tested again,* and if it's true, the loop body is executed starting from the top.
  - In a do/while loop, execution skips to the bottom of the loop, where the loop condition is tested again before restarting the loop at the top.
  - In a for loop, the *increment expression is evaluated, and the test expression is* tested again to determine if another iteration should be done.

Creating Objects in Javascript

- Objects can be created with *object literals*, with the *new* keyword, and with the *Object.create()* function.
- Object Literals: An *object literal is a comma-separated list of colon-separated name:value pairs, enclosed* within curly braces.

- A property name is a JavaScript identifier or a string literal (the empty string is allowed). A property value is any JavaScript expression; the value of the expression (it may be a primitive value or an object value) becomes the value of the property.

e.g. var point = { x:0, y:0 };

Creating Objects in Javascript

- new keyword: An *object literal is a comma-separated list of colon-separated name:value pairs, enclosed* within curly braces.

- The new operator creates and initializes a new object. The new keyword must be followed by a function invocation.

- A function used in this way is called a *constructor and serves* to initialize a newly created object.

Creating Objects in Javascript

E.g.

```
var o = new Object(); // Create an empty object: same as {}.
var a = new Array(); // Create an empty array: same as [].
var d = new Date(); // Create a Date object representing the current time
var r = new RegExp("js"); // Create a RegExp object for pattern matching.
```

Creating Objects in Javascript

- Prototypes:
  - Every JavaScript object has a second JavaScript object associated with it. This second object is known as a prototype, and the first object inherits properties from the prototype.
  - All objects created by object literals have the same prototype object, and we can refer to this prototype object in JavaScript code as Object.prototype.
  - Objects created using the new keyword and a constructor invocation use the value of the prototype property of the constructor function as their prototype.
  - So the object created by new Object() inherits from Object.prototype just as the object created by {} does.

Creating Objects in Javascript

- Object.create() :
  - It creates a new object, using its first argument as the prototype of that object.
  - It also takes an optional second argument that describes the properties of the new object.

E.g. var o1 = Object.create({x:1, y:2}); // o1 inherits properties x and y.

  - If you want to create an ordinary empty object (like the object returned by {} or new Object()), pass Object.prototype:

e.g. var o3 = Object.create(Object.prototype); // o3 is like {} or new Object().

Querying & Setting Properties in Javascript

- To obtain the value of a property, use the dot (.) or square bracket ([]) operators.
- The left-hand side should be an expression whose value is an object.
- If using the dot operator, the right-hand must be a simple identifier that names the property.

Querying & Setting Properties in Javascript

- If using square brackets, the value within the brackets must be an expression that evaluates to a string that contains the desired property name:

var author = book.author; // Get the "author" property of the book.

var name = author.surname // Get the "surname" property of the author.

var title = book["main title"] // Get the "main title" property of the book.

Querying & Setting Properties in Javascript

- To create or set a property, use a dot or square brackets as you would to query the property, but put them on the left-hand side of an assignment expression:

book.edition = 6; // Create an "edition" property of book.

book["main title"] = "ECMAScript"; // Set the "main title" property.

Property getters & setters in Javascript

- An object property is a name, a value, and a set of attributes, the value may be replaced by one or two method as a *getter* and a *setter.*

- Properties defined by getters and setters are sometimes known as *accessor properties to distinguish them from data properties that have a simple value.*