Client Side Scripting Language (22519)

THEORY PAPER	: 70
PROGRESSIVE ASSESSMENT	: 30 (PTT + MICRO PROJECT)
TERM WORK	: 25
EXTERNAL PRACTICAL EXAM	: 25

Chapter 1: Basics of Javascript Programming (12 marks)

Features of JavaScript, Object Name, Property, method, Dot syntax, main event. Values and Variables

Operators and Expressions- Primary Expressions, Object and Array initializers,

function definition expression, property access expressions, invocation expressions.

If Statement, if...else, if..elseif, nested if statement.

Switch...case statement

Loop statement — for loop, for...in loop, while loop, do...while loop, continue Statement.

Querying and setting properties and deleting properties, property getters and setters.

Features of JavaScript

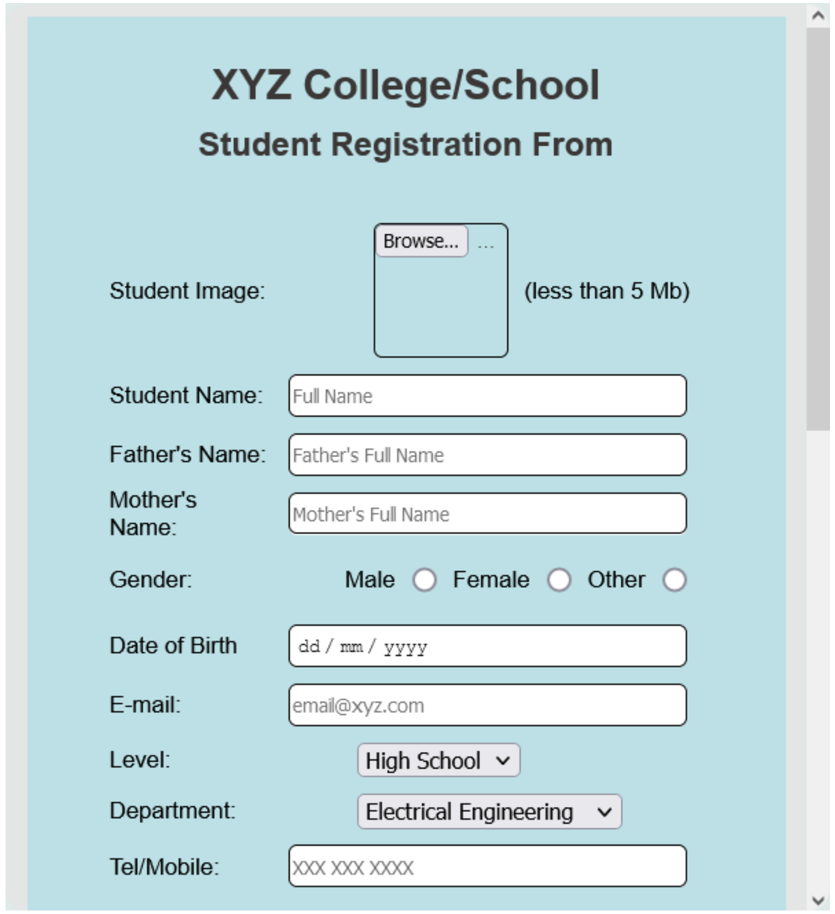
- Build Dynamic Web Pages
- Display Alert Boxes
- Write messages to the browser status bar
- Control features of the browser
- Perform Calculations
- Customize reactions to mouse actions and keystrokes
- Open new browser Windows
- Validation Information in forms

Basic Concepts in JS

JavaScript is an object-oriented programming language that is used to build programs using objects.

In programming, the objects most commonly used by JavaScript are

- Documents
- Forms
- Fields
- radio buttons
- other elements that you find on a form or user interface.
- A window is also an object used by a JavaScript program



XYZ College/School

Student Registration From

Student Image:	Browse...	(less than 5 Mb)
Student Name:	Full Name	
Father's Name:	Father's Full Name	
Mother's Name:	Mother's Full Name	
Gender:	Male ○ Female ○ Other ○	
Date of Birth	dd / mm / yyyy	
E-mail:	email@xyz.com	
Level:	High School	
Department:	Electrical Engineering	
Tel/Mobile:	XXX XXX XXXX	

Object Name, Property, Methods, Dot Syntax

Object Name

Each object must be uniquely identified by a name or ID that you assign to the object to reference it from your JavaScript. Forms, for example, could be named form1 and form2. Alternatively, you could assign forms names that identify the purpose of each form, such as *OrderEntryForm* and *OrderDisplayForm*, which more clearly identify each form in your JavaScript.

Object Name, Property, Methods, Dot Syntax Continue ....

Property

A property is a value that is associated with an object. Objects can have many values.

E.g.

- A window has a background colour, a width, and height.
- a form object has a title, a width, and a height.

Object Name, Property, Methods, Dot Syntax Continue ….

Methods

A method is a process performed when a event performed on object.

E.g.

- For example, a Submit button on a form is an object. Its Submit label and the dimensions of the button are properties of the button object. If you click the Submit button, the form is submitted to the server-side application. In other words, clicking the Submit button causes the button to process a method.

Object Name, Property, Methods, Dot Syntax Continue ….

Dot Syntax

A method is a process performed when a event performed on object.

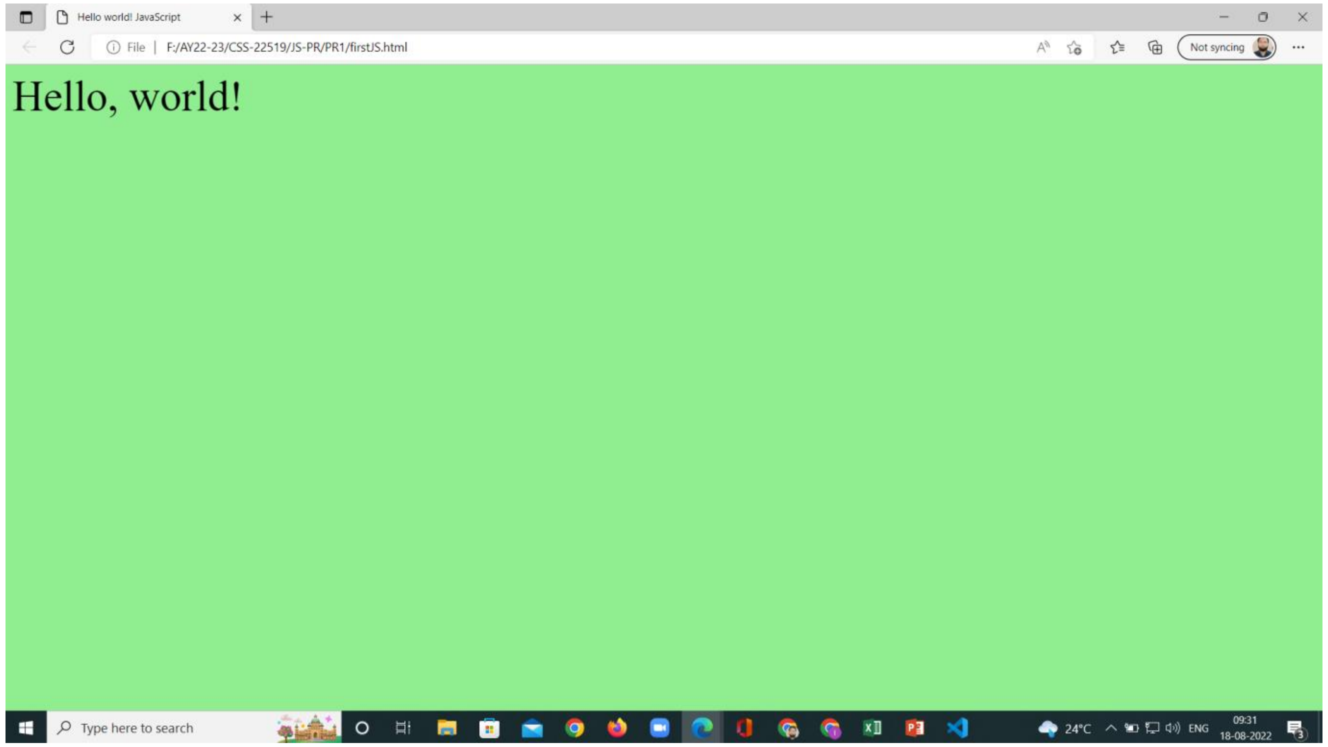
E.g. For example, a document is an object that has a certain background colour (property) and that can be written to (method). You access an object's properties and methods by using the dot syntax along with the object name and its property or method. So, for example, here's how you would identify the background colour of a document and the write method for a document:

document . bgColor

document . write()

Your First JavaScript

```html
<html >

    <head>

        <title>Hello world! JavaScript</title>

    </head>

    <body>

        <p style="font-size: 50px;">

        <script language="Javascript" type="text/javascript">

            document.write('Hello, world!')

            document.bgColor = 'lightgreen';

        </script>

        </p>

    </body>

</html>
```



```
Hello, world!
```

Operators and Expressions

An ***expression*** is a valid unit of code that resolves to a value.

A mathematical expression consists of two parts: operands and operators. An operand is the value.

An ***operator*** is the symbol that tells the browser how to evaluate the mathematical expression.

1. Arithmetic operators
2. Logical operators.
3. Assignment operators.
4. Comparison operators.
5. Conditional operators.

Expressions

Any unit of code that can be evaluated to a value is an expression. Since expressions produce values, they can appear anywhere in a program where JavaScript expects a value such as the arguments of a function invocation.

Arithmetic Expressions:

Arithmetic expressions evaluate to a numeric value. Examples include the following

```
10;    // Here 10 is an expression that is evaluated to the numeric value 10 by the JS interpreter

10+13; // This is another expression that is evaluated to produce the numeric value 23
```

Array and object initializers expressions

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.

Some Examples are:

```
[]              // array literal with no element
{}              // object literal with no property
[1,2,3]         // A 3-element array
{a: 1, b: 2}    // An object with 2 properties
{a: {b: 1}}     // Nested Object literals
```

Function definition expressions

A function definition expression defines a JavaScript 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.

Some examples are:

```
function() { }                  // function definition without arguments
function(a, b) { return a * b }   // function definition with arguments
f1 = (a, b) => a * b             // arrow function : to define short definitions with arguments
() => { return 2 }               // arrow function : to define short definitions without arguments
```

Property access expressions

A property access expression evaluates to the value of an object property or an array element. JavaScript defines two syntaxes for property access:

- expression . Identifier
- expression [expression]

In first style of property access, expression specifies the object, and the identifier specifies the name of the desired property.

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 or the index of the desired array element.

Example:

```
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
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]
```

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.

Some examples:

```
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.
```

Arithmetic Operators

Operator	Operation	Example	Operation Result
+	Addition	Z = 10 + 12	Z = 22
-	Subtraction	Z = 10 - 12	Z = -2
*	Multiplication	Z = 10 * 12	Z = 120
/	Division	Z = 10 / 12	Z = 0.83

%	Modulus	X = 23 % 10
		Z = 7 % 10
			X = 3
			Z = 7

++	Increment by 1	a = 10
		b = ++a (increment and then assigned)
		c = a++ (Assigned first then increment)
			
			b = 11
			c = 11

--	Decrement by 1	a = 10
		b = --a (decrement and then assigned)
		c = a-- (Assigned first then decrement)
			
			b = 9
			c = 9

Logical Operators continue…

Operator	Operation	Example	Operation Result

&&	Logical AND	userID = ‘C1002’
		Password = ‘P1002’
		userID == ‘C1002’ && password == ‘P1002’
			TRUE

||	Logical OR	userID = ‘C1002’
		Password = ‘P1003’
		userID == ‘C1002’ || password == ‘P1002’
			TRUE

!	Logical NOT	userID = ‘C1002’ ! userID == ‘C1002’	FALSE

Logical Operators

Logical operators are *used to combine two logical expressions* into one expression.

A logical expression is an expression that evaluates to either true or false.

- E.g. *Is userID equivalent to C1002 ? (result either* TRUE *or* FALSE*)*

  userID == 'C1002' (logical expression written in JS)

  Is password equivalent to P1002 ? (result either TRUE *or* FALSE*)*

  password == 'P1002' (logical expression written in JS)

We can combine above two logical expressions in to one by using logical operators.

Assignment Operator

Operator	Operation	Example	Equivalent Expression
=	Assign	A =20	
+=	Add value then assign	A += 20	A = A + 20
-=	Subtract value then assign	A -= 20	A = A - 20
*=	Multiply value then assign	A *= 20	A = A * 20
/=	Divide value then assign	A /= 20	A = A / 20
%=	Modulus value then assign	A %= 20	A = A % 20

Comparison Operators

Operator	Operation	Example	Equivalent Expression
= =	Equal to	'20' == 20	Will return TRUE if values are equal, irrespective of datatype. '20' == 20 will return TRUE.
===	Equal to (strict type)	'20' === 20	Will return TRUE if values as well datatype are equal. '20'===20 will return FALSE.
>	Greater than	A > 20	Will return TRUE if A's value is greater than 20.
<	Less than	A < 20	Will return TRUE if A's value is less than 20.
>=	Greater than equal to	A >= 20	Will return TRUE if A's value is greater than or equal to 20.
<=	Less than equal to	A <= 20	Will return TRUE if A's value is less than or equal to 20.
! =	Not equal to	A != 20	Will return TRUE if A's value is not equal to 20.

Conditional Operator

The conditional operator has three parts,

The first part is a logical expression, which you'll recall is an expression that evaluates to either true or false.

The second part is the action the browser must take if the expression is true.

The third part is the action the browser must take if the expression is false.

The fi rst and second parts of the conditional operator are separated by a question mark (?).

The second part and the third parts are separated by a colon (:).

Example:

```
userID == 'ScubaBob' && password == 'diving' ? message = 'Approved' : message = 'Rejected‘
```

Conditional statements

Conditional statements execute or skip other statements depending on the value of a specified expression. These statements are the decision points of your code.

1. if statement
2. If-else statement
3. If-else-if statement
4. switch statement

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 for single statement execution whenever expression is true

Syntax:

```
if (expression)
        statement
```

Example:

```
if (username == null)
        console.log(“Must Enter Username”)
```

The second is for multiple statement execution whenever expression is true.

Syntax:

```
if (expression)
        {
        statement 1
        statement 2
        :
        }
```

Example

```
if (expression)
{
address = "";
message = "Please specify a mailing address.";
}
```

If-else statement

If-else statement introduces an else block that is executed when expression is false and if block executed when expression is true.

Syntax:

```
if (expression)

{ if block statements
}
else
{
else block statements
}
```

The second is for multiple statement execution whenever expression is true.

Syntax:

```
if (expression)
         {
         statement 1
         statement 2
         :
         }
```

Example

```
if (expression)
{
address = "";
message = "Please specify a mailing address.";
}
```

If-else-if statement

If-else statement introduces an else block that is executed when expression is false and if block executed when expression is true.

Syntax:

```
if (expression)

          { if block statements
          }
else if (expression)
          { else if block statements
          }
          :
          :
else
          {
          else block statements
          }
```

Example:

```
avg = 74
if (avg < 40) {
           console.log(‘F’)
          }
else if (avg < 50) {
           console.log(‘D’)
          }
else if (avg < 60) {
           console.log(‘C’)
          }
else if (avg < 75) {
            console.log(‘B’)
          }
else      { console.log(‘A’)
          }
```

switch statement

Syntax:

```
switch(n)

{ case 1:  // Start here if n == 1

           // Execute code block #1.

           break; // Stop here

case 2: // Start here if n == 2

           // Execute code block #2.

           break; // Stop here

case 3: // Start here if n == 3

           // Execute code block #3.

           break; // Stop here

default: // If all else fails...

           // Execute code block #4.

           break; // stop here }
```

Syntax:

```
switch(typeof x)

{

case 'number':

          return x.toString(16);
          break;

case 'string':

          return '"' + x + '"';
           break;

default:

          return String(x);
          break;

}
```

Loops

The looping statements are those that repeat portions of your code.

JavaScript has four looping statements: while, do/while, for, and for/in. The subsections below explain each in turn.

1. for loop
2. while loop
3. do-while loop
4. for-in loop

while Loop

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.
Another way to say this is that the interpreter executes statement repeatedly while the expression is true.
Note that you can create an infinite loop with the syntax `while(true)`.

Syntax:

```
while(expression)
{ statements
}
```

Example:

```
var count = 0;
while (count < 10)
{
console.log(count);
count++;
}
```

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.

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

Syntax:

```
do
{ statements
} while(expression);
```

Example:

```
a = [1,2,3,4,5]
var len = a.length, i = 0;
if (len == 0)
          console.log("Empty Array");
else {
          do
          { console.log(a[i]);
          } while (++i < len);
     }
```

for Loop

The for statement provides a looping construct that is often more convenient 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. The for statement encodes each of these three manipulations as an expression and makes those expressions an explicit part of the loop syntax:

Syntax:

```
for(initialize ; test ; increment)

          {  statement

          }
```

Example 1:

```
for(var count = 0; count < 10; count++)
          console.log(count);
```

Example 2:

```
for(i = 0, j = 10 ; i < 10 ; i++, j--)

          {        sum += i * j;

          }
```

for loop continue ....

In all above loop examples so far, the loop variable has been numeric. This is quite common but is not necessary.

The following code uses a for loop to traverse a linked list data structure and return the last object in the list (i.e., the first object that does not have a next property):

```
function tail(o) {                                      // Return the tail of linked list o
            for(; o.next; o = o.next) /* empty */ ;     // Traverse while o.next is truthy
            return o;
            }
```

Note that
If you omit the test expression, the loop repeats forever, and `for(;;)` is another way of writing an infinite loop, like `while(true)`.

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. Regular for loop terminates when a condition expression results false,
but 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 outside the loop.

If object expression is not `null` or `undefined`, the interpreter now executes the body of the loop once for each enumerable property of the object. Before each iteration, however, the interpreter evaluates the variable expression and assigns the **`name of the property (a string value)`** to it.

Syntax:

```
for (variable in object)
          statement
```

Example 1:

```
//Write JS program to copy property names of object o in the array a
          var o = {x:1, y:2, z:3};
          var a = [], i = 0;
          for(a[i++] in o)
                    console.log(a)
```

Output:

```
[ 'x' ]
[ 'x', 'y' ]
[ 'x', 'y', 'z' ]
```

for-in Loop

Example 2:

```
var a = ['a','b','c','d']

for(i in a)

        console.log(i)
```

Output:

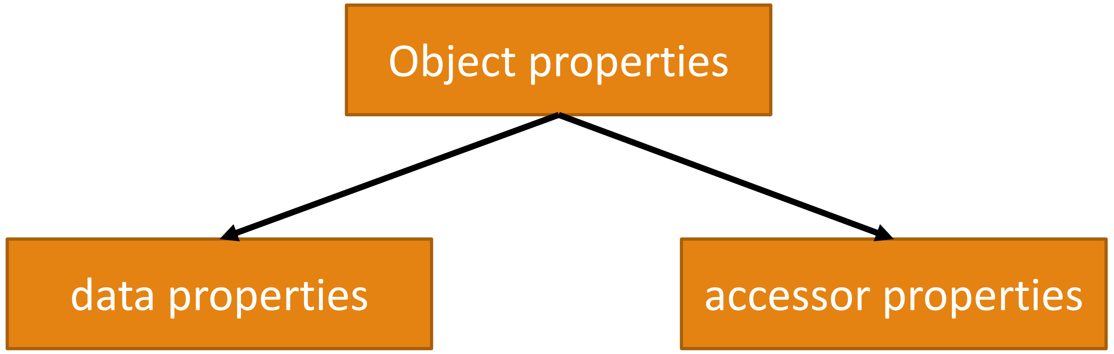
```
0

1

2

3
```

Note: We printed i value inside the loop, and can observe that values are 0 1 2 3, so we can say that the index values are assigned as property names to i in every iteration.

Property Getters and Setters

Object property is a name, a value, and a set of attributes. In ECMAScript 5 the value may be replaced by one or two methods, known 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.



Property Getters and Setters continue….

When a program queries (access) the value of an accessor property, JavaScript invokes the getter method (passing no arguments). The return value of this method becomes the value of the property access expression.

When a program sets (change) the value of an accessor property, JavaScript invokes the setter method, passing the value of the right-hand side of the assignment. This method is responsible for “setting,” in some sense, the property value. The return value of the setter method is ignored.

In example, object **`o`** declared with two properties **`x (data property)`** and **`r (accessor property)`**.

```
var o =

{

         x:2 ,

         get r(){return this.x * this.x} ,

         set r(v){return this.x=v}

};

console.log(o.r)  // accessing a property r, so getter method will be invoked

o.r = 3      // changing a property r, so setter method will be invoked with arg 3

console.log(o.r) //accessing a property r so getter method will be invoked
```

Output:

```
4

9
```