TechLift
KICKSTART YOUR CAREER

WEB ENGINEERING

.Net

P@SHA

PAKISTAN PSEB SOFTWARE EXPORT BOARD

Week 1 - Day 4 & 5

Learning Objectives

By the end of this session, the students will have reinforced to further developed an understanding of:

- Functions
- Data Types
- Operators and Arithmetic Operations
- Error Handling

: A Quick Review

This week is focused on everyone (students with technical/non technical background) starting from the same ground with HTML & JavaScript revisions.

Reinforcement

- What is Javascript?
- What are pros and cons of Javascript Language?
- what can you do with JavaScript?
- What are the types of Javascript?
- What are the output ways?
- How do you declare variables?
- What are the types of statements?

Statements

Statements

JavaScript statements are composed of:

- Values
- Operators
- Expressions
- Keywords, and
- Comments

Order of execution of Statements is the same as they are written.

Semicolons:

- Semicolons separate JavaScript statements.
- Semicolon marks the end of a statement in javascript

Statements

Statements also very similar to Java

Conditional	Loops	Exceptions
if(){}else{}	while(){};	throw
switch(){}	do{}while();	try{}catch(){};
	for(){};	
	continue;	
	break	

JavaScript - Loops

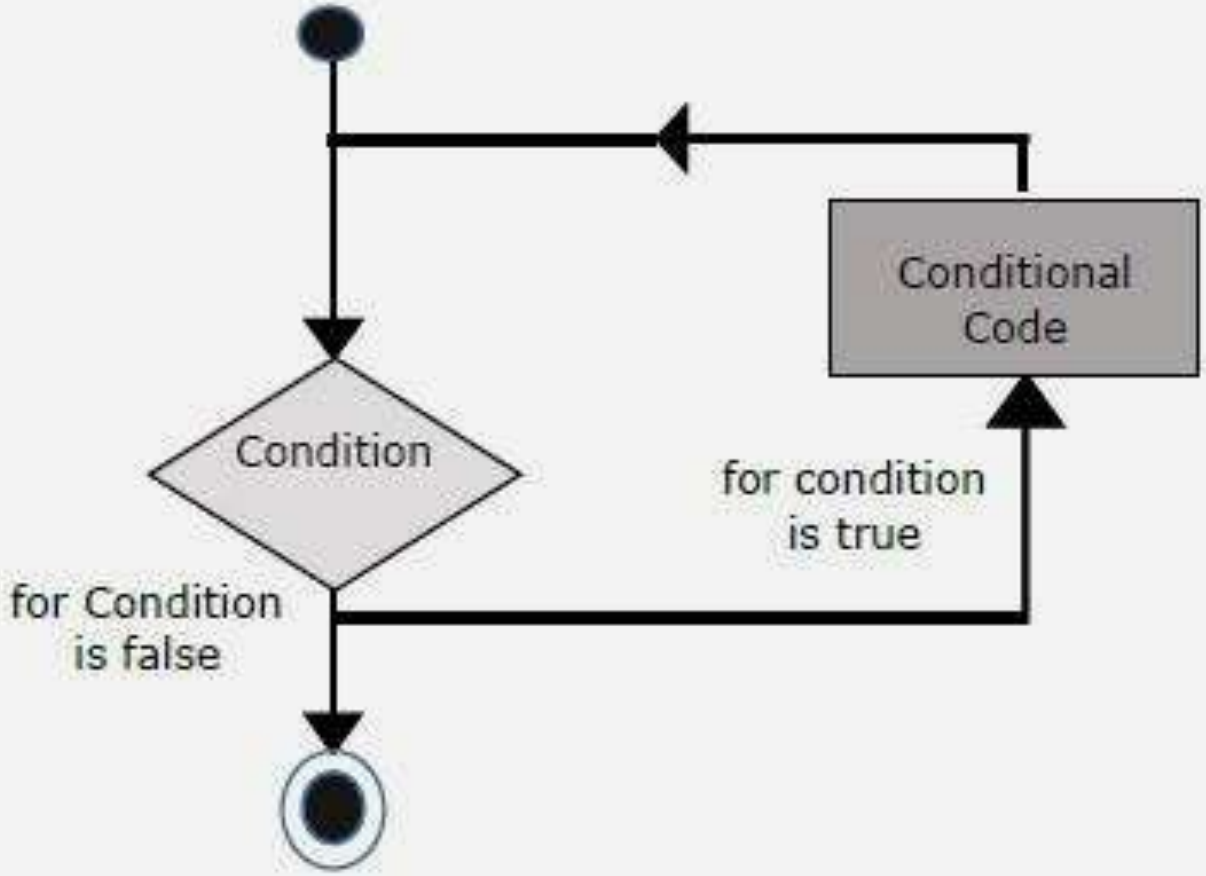
- The JavaScript loops are used to iterate the piece of code using for, while, do while or for-in loops. It makes the code compact. It is mostly used in array.
- There are the following types of loops in JavaScript.
  - for loop
  - while loop
  - do-while loop
  - for-in loop
  - continue
  - break

1- JavaScript For loop

- The '**for**' loop is the most compact form of looping. It includes the following three important parts −
- The **loop initialization** where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The t**est statement** which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The **iteration statement** where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

1- JavaScript For loop



1- JavaScript For loop

The syntax of **for** loop is JavaScript is as follows –

```
for (initialization; test condition; iteration statement) {
Statement(s) to be executed if test condition is true
}
```

1- JavaScript For loop - Example

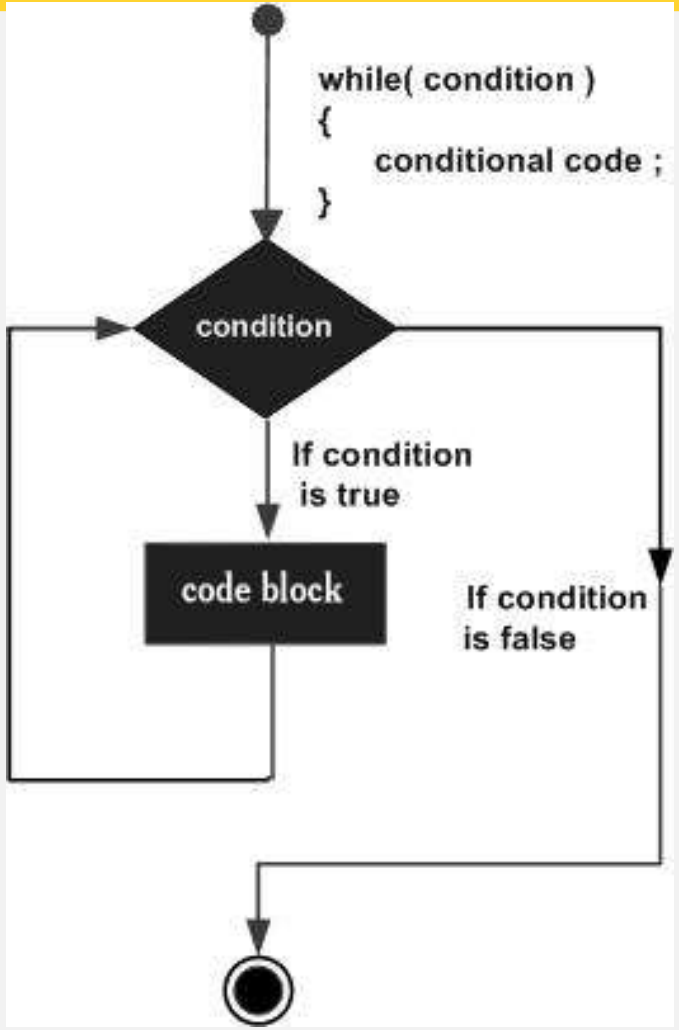
Try it yourself. Then tally your output with the one given on the next slide:

Check the output here.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var count;
            document.write("Starting Loop" + "<br />");

            for(count = 0; count < 10; count++) {
               document.write("Current Count : " + count );
               document.write("<br />");
            }
            document.write("Loop stopped!");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

2- JavaScript while loop

The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.



FlowChart

The flowchart of while loop looks as follows −

2- JavaScript while loop - Syntax

The syntax of while loop in JavaScript is as follows −

```
while (expression) {
Statement(s) to be executed if expression is true
}
```

2- JavaScript while loop

Let’s see the simple example of while loop in javascript.

Check the output here.

```
<html>
   <body>

      <script type = "text/javascript">
         <!--
            var count = 0;
            document.write("Starting Loop ");

            while (count < 10) {
               document.write("Current Count : " + count + "<br />");
               count++;
            }

            document.write("Loop stopped!");
         //-->
      </script>

      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

3- JavaScript do while loop

The JavaScript do while loop iterates the elements for the infinite number of times like while loop. But, code is executed at least once whether condition is true or false. The syntax of do while loop is given below.

```
do{
    code to be executed
}while (condition);
```

3- JavaScript do while loop

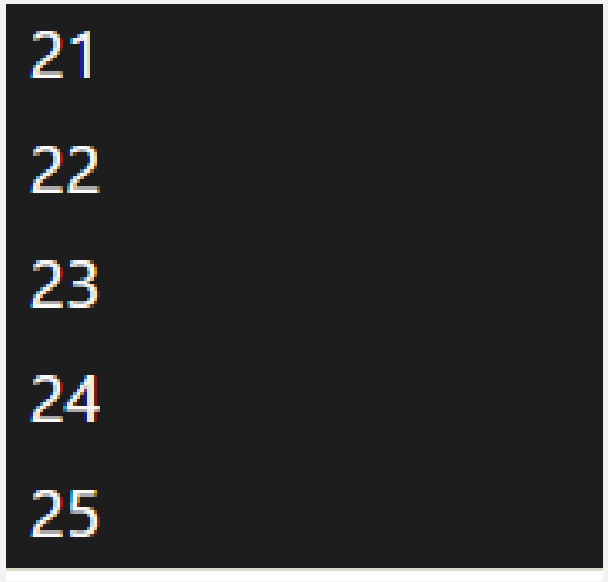
Let’s see the simple example of do while loop in javascript.

Try it yourself:

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}while (i<=25);
</script>
```

3- JavaScript do while loop

Output



4- JavaScript for in loop - Example

Try the following example to implement ‘for-in’ loop. It prints the web browser Navigator object.

Check the result here.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var aProperty;
            document.write("Navigator Object Properties<br /> ");
            for (aProperty in navigator) {
               document.write(aProperty);
               document.write("<br />");
            }
            document.write ("Exiting from the loop!");
         //-->
      </script>
      <p>Set the variable to different object and then try...</p>
   </body>
</html>
```

4- JavaScript for in loop

The **for...in** loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

The syntax of 'for..in' loop is –

```
for (variablename in object) {
statement or block to execute
}
```

In each iteration, one property from **object** is assigned to **variablename** and this loop continues till all the properties of the object are exhausted.

5- JavaScript - Loop Control

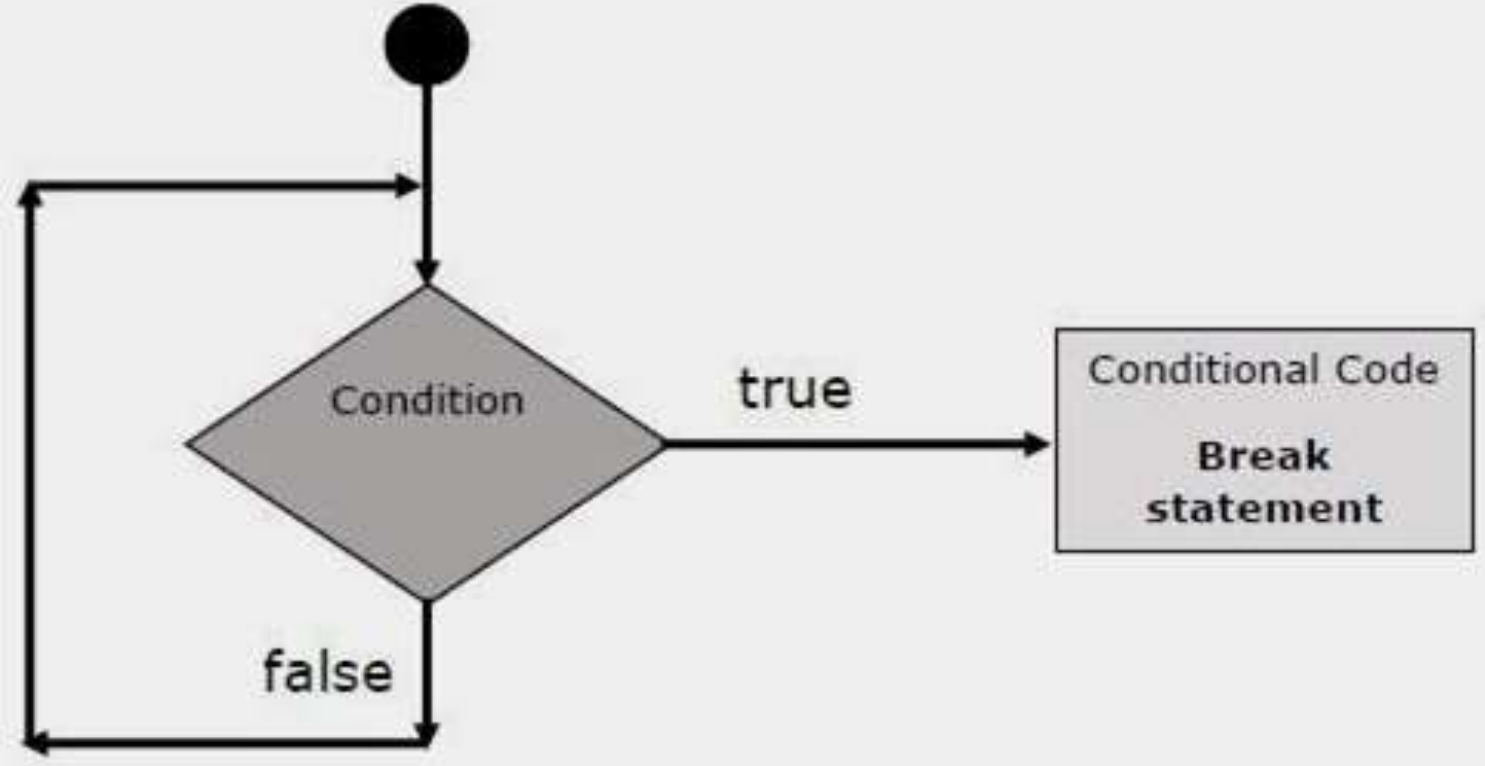
- JavaScript provides full control to handle loops and switch statements.
- There may be a situation when you need to come out of a loop without reaching its bottom.
- There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

- To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The break statement, which was briefly introduced with the switch statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart
The flow chart of a break statement would look as follows −



The break Statement - Example

The following example illustrates the use of a break statement with a while loop. Notice how the loop breaks out early once x reaches 5 and reaches to document.write (..) statement just below to the closing curly brace −.

Check output here.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
         var x = 1;
         document.write("Entering the loop<br /> ");

         while (x < 20) {
            if (x == 5) {
               break;   // breaks out of loop completely
            }
            x = x + 1;
            document.write( x + "<br />");
         }
         document.write("Exiting the loop!<br /> ");
         //-->
      </script>

      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

The Continue Statement

- The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block.
- When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

The Continue Statement

This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 −

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var x = 1;
            document.write("Entering the loop<br /> ");

            while (x < 10) {
               x = x + 1;

               if (x == 5) {
                  continue;   // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

The Continue Statement

This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5 −

Check the output here.

```
<html>
   <body>
      <script type = "text/javascript">
         <!--
            var x = 1;
            document.write("Entering the loop<br /> ");

            while (x < 10) {
               x = x + 1;

               if (x == 5) {
                  continue;   // skip rest of the loop body
               }
               document.write( x + "<br />");
            }
            document.write("Exiting the loop!<br /> ");
         //-->
      </script>
      <p>Set the variable to different value and then try...</p>
   </body>
</html>
```

Task: Create a Simple Timer

Create the timer by watching this video.

Functions

Functions

- Fundamental building blocks in JavaScript
- Perform a set of statements
  - Calculate a value
  - Perform a task
- Define on the scope from where you want to call it

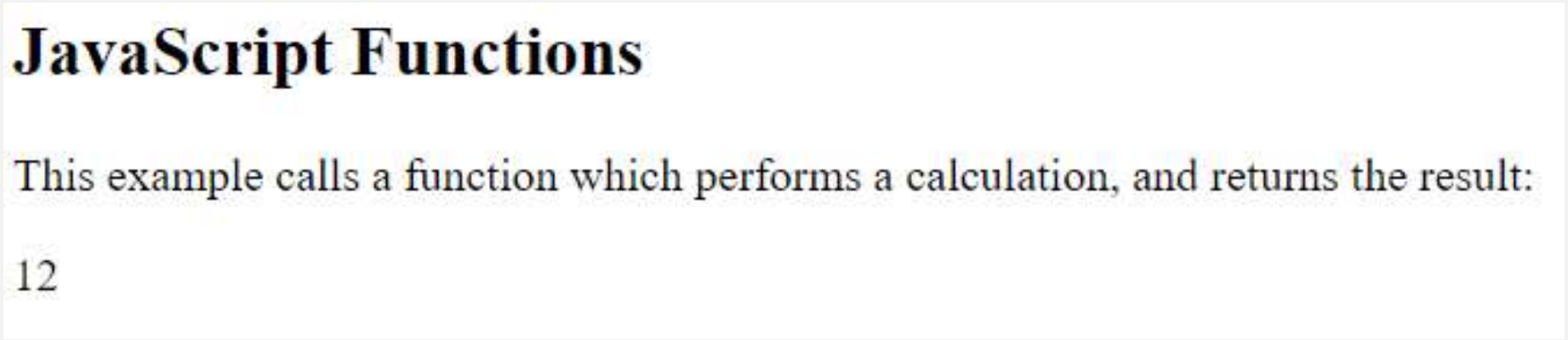
Functions

- Fundamental building blocks in JavaScript to perform a particular task
- Perform a set of statements
  - Calculate a value
  - Perform a task
- Define on the scope from where you want to call it

(A JavaScript function is executed when "something" invokes it (calls it).)

Functions - Example

Write the code to get the following output and then check the code from the link given below:



Check the output here.

JavaScript Function Syntax

- A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:

  (parameter1, parameter2, ...)

- The code to be executed, by the function, is placed inside curly brackets**: {}**

JavaScript Function Syntax

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.
- A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Function Invocation

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- Function parameters are listed inside the parentheses () in the function definition.
- Function arguments are the values received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.
- A Function is much the same as a Procedure or a Subroutine, in other programming languages.

Functions

Syntax

```
<script type = "text/javascript">

 function functionname(parameter-list)

{ statements }

 </script>
```

Example

```
<script type ="text/javascript">
 function sayHello()
      {alert("Hello there"); }
 </script>
```

Calling a Function

```
<html>  <head>
        <script type = "text/javascript">
                function sayHello()
                        {       document.write ("Hello there!");       }
        </script>   </head>
 <body>
  <p>Click the following button to call the function</p>
      <form>       <input type = "button" onclick = "sayHello()" value = "Say Hello">     </form>
   <p>Use different text in write method and then try...</p>
   </body>
</html>
```

The return Statement

This is required to return a value from a function. This statement should be the last statement in a function.

```
<html>
 <head>
      <script type = "text/javascript">
               function concatenate(first, last)
                    {
                              var full;
                              full = first + last;
                              return full;
                    }
               function secondFunction()
                    {         var result;
                              result = concatenate('snehal', 'smruti');
                              document.write (result );
                    }
          </script>
   </head>
 <body>
    <p>Click the following button to call the function</p>
       <form>
              <input type = "button" onclick = "secondFunction()" value = "Call Function">
        </form>

   </body>
</html>
```

Recap of Day 4

The trainer will play a musical cusion/book/paper ball segment to quickly recap day 4. The participants where the clip will stop will quickly recap a topics from the ones listed below:

1. The Return Statement
2. JavaScript Function Syntax
3. The Return Statement

Day 5

Data Types

Data Types

There are two types of Data Types in JavaScript

– Primitive data type

– Non-primitive (reference) data type

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

Primitive data type

- String
- Number
- Boolean
- Null
- Undefined

Primitive data type (String)

A series of characters enclosed in quotation marks either single quotation marks (') or double quotation marks (")

Example :

```
var name = “Webstack Academy”;
var name = 'Webstack Academy';
```

JavaScript Booleans

Booleans can only have two values: true or false. Booleans are often used in conditional testing.

Example

```
let x = 5;
let y = 5;
let z = 6;
(x == y)       // Returns true
(x == z)       // Returns false
```

Execute the example and check the output here.

JavaScript Arrays

- JavaScript arrays are written with square brackets.
- Array items are separated by commas.
- The following code declares (creates) an array called cars, containing three items (car names):

Example

```
const cars = ["Saab", "Volvo", "BMW"];
```

check the output here.

Array indexes are zero-based, which means the first item is [0], second is [1], and so on.

You will learn more about arrays later in this tutorial.

JavaScript Objects

- JavaScript objects are written with curly braces {}.
- Object properties are written as name:value pairs, separated by commas.

Example

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Check the output here.

The object (person) in the example above has 4 properties: firstName, lastName, age, and eyeColor.

Undefined

In JavaScript, a variable without a value, has the value undefined. The type is also undefined.

Example:

```
let car;    // Value is undefined, type is undefined
```

Check the output here.

Undefined

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

Example:

```
car = undefined;     // Value is undefined, type is undefined
```

Check the output here.

Operators

Primitive Data Type (Numbers)

JavaScript has only one type of numbers.

Numbers can be written with, or without decimals:

Example

```
let x1 = 34.00;      // Written with decimals
let x2 = 34;         // Written without decimals
```

Execute the example and check the output here.

Operators

JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators

Let's have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first Ex: A - B will give -10
3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2

Arithmetic Operators

Assume variable A holds 10 and variable B holds 20, then −

Sr.No.	Operator & Description
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Arithmetic Operators

Practice Task

Try writing the code given in the example here.
After writing it, check the output.

Comparison Operators

The JavaScript comparison operator compares the two operands. The comparison operators are as follows:

Operator	Description	Example
==	Is equal to	10==20 = false
===	Identical (equal and of same type)	10==20 = false
!=	Not equal to	10!=20 = true
!==	Not Identical	20!==20 = false
>	Greater than	20>10 = true
>=	Greater than or equal to	20>=10 = true
<	Less than	20<10 = false
<=	Less than or equal to	20<=10 = false

Comparison Operators

Try writing the code given in the example here.
After writing it, check the output.

Practice Task

Logical Operators

The following operators are known as JavaScript logical operators.

Operator	Description	Example
&&	Logical AND	(10==20 && 20==33) = false
||	Logical OR	(10==20 || 20==33) = false
!	Logical Not	!(10==20) = true

Logical Operators

Try writing the code given in the example here.
After writing it, check the output.

Practice Task

Assignment Operators

The following operators are known as JavaScript assignment operators.

Operator	Description	Example
=	Assign	10+10 = 20
+=	Add and assign	var a=10; a+=20; Now a = 30
-=	Subtract and assign	var a=20; a-=10; Now a = 10
*=	Multiply and assign	var a=10; a*=20; Now a = 200
/=	Divide and assign	var a=10; a/=2; Now a = 5
%=	Modulus and assign	var a=10; a%=2; Now a = 0

Assignment Operators

Try writing the code given in the example here.
After writing it, check the output.

Practice Task

Errors & Exceptions Handling

Syntax Errors

Find out the reason that why does the following line causes a syntax error:

```
<script type = "text/javascript">
<!--
window.print(;
//-->
</script>
```

Runtime Errors

Find out the reason that why does the following line causes a runtime error

```
<script type = "text/javascript">
<!--
window.printme();
//-->
</script>
```

Logical Errors: The try...catch...finally Statement

1

Here is an example where we are trying to call a non-existing function which in turn is raising an exception. Let us see how it behaves without **try...catch−**

Try writing this code and then tally your output.

Check the output here.

```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               alert("Value of variable a is : " + a );
            }
         //-->
      </script>
   </head>

   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>
   </body>
</html>
```

2

Now let us try to catch this exception using t**ry...catch** and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

Try writing this code and then tally your output.

Check the output here.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               try {
                  alert("Value of variable a is : " + a );
               }
               catch ( e ) {
                  alert("Error: " + e.description );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

3

You can use finally block which will always execute unconditionally after the try/catch. Here is an example.

Try writing this code and then tally your output.

Check the output here.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;

               try {
                  alert("Value of variable a is : " + a );
               }
               catch ( e ) {
                  alert("Error: " + e.description );
               }
               finally {
                  alert("Finally block will always execute!" );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

The Throw Statement

- You can use throw statement to raise your built-in exceptions or your customized exceptions.
- Later these exceptions can be captured and you can take an appropriate action.

The Throw Statement

This example demonstrates how to use a throw statement.

Try writing this code and then tally your output.

Check the output here.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               var b = 0;

               try {
                  if ( b == 0 ) {
                     throw( "Divide by zero error." );
                  } else {
                     var c = a / b;
                  }
               }
               catch ( e ) {
                  alert("Error: " + e );
               }
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

P@SHA PSEB

The onerror() Method

- The onerror event handler was the first feature to facilitate error handling in JavaScript.
- The error event is fired on the window object whenever an exception occurs on the page.

The onerror() Method

Try writing this code and then tally your output.

Check the output here.

```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            window.onerror = function () {
               alert("An error occurred.");
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

The onerror() Method

- The **onerror** event handler provides three pieces of information to identify the exact nature of the error −
- **Error message** − The same message that the browser would display for the given error
- **URL −** The file in which the error occurred
- **Line number**− The line number in the given URL that caused the error

The onerror() Method

Here is the example to show how to extract this information.

Try writing this code and then tally your output.

Check the output here.

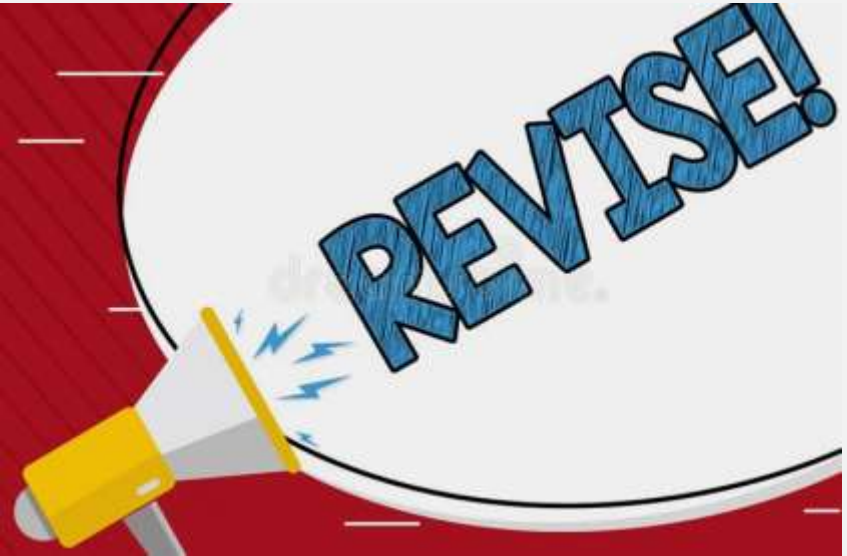
```
<html>
   <head>

      <script type = "text/javascript">
         <!--
            window.onerror = function (msg, url, line) {
               alert("Message : " + msg );
               alert("url : " + url );
               alert("Line number : " + line );
            }
         //-->
      </script>

   </head>
   <body>
      <p>Click the following to see the result:</p>

      <form>
         <input type = "button" value = "Click Me" onclick = "myFunc();" />
      </form>

   </body>
</html>
```

Homework

1- Students will practice exercises of the topics covered today from this link:

JavaScript Exercises

2- They will develop a multiplication table with the help of this link.



They will come prepared for a marked quiz to be held in the next class.

Reinforcement Tutorial

This tutorial can be shared with the students for further practice and reinforcement: https://www.guru99.com/interactive-javascript-tutorials.html

Learning Objectives

By the end of this session, the students have practised

- Functions
- Data Types
- Operators and Arithmetic Operations
- Error Handling

Conclusion & Q/A

See you tomorrow!