JAVASCRIPT - ERRORS & EXCEPTIONS HANDLING

There are three types of errors in programming: *a* Syntax Errors, *b* Runtime Errors, and *c* Logical Errors.

Syntax Errors

Syntax errors, also called **parsing errors,** occur at compile time in traditional programming languages and at interpret time in JavaScript.

For example, the following line causes a syntax error because it is missing a closing parenthesis.

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

When a syntax error occurs in JavaScript, only the code contained within the same thread as the syntax error is affected and the rest of the code in other threads gets executed assuming nothing in them depends on the code containing the error.

Runtime Errors

Runtime errors, also called **exceptions,** occur during execution *aftercompilation/interpretation*.

For example, the following line causes a runtime error because here the syntax is correct, but at runtime, it is trying to call a method that does not exist.

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

Exceptions also affect the thread in which they occur, allowing other JavaScript threads to continue normal execution.

Logical Errors

Logic errors can be the most difficult type of errors to track down. These errors are not the result of a syntax or runtime error. Instead, they occur when you make a mistake in the logic that drives your script and you do not get the result you expected.

You cannot catch those errors, because it depends on your business requirement what type of logic you want to put in your program.

The try...catch...finally Statement

The latest versions of JavaScript added exception handling capabilities. JavaScript implements the **try...catch...finally** construct as well as the **throw** operator to handle exceptions.

You can **catch** programmer-generated and **runtime** exceptions, but you cannot **catch** JavaScript syntax errors.

Here is the **try...catch...finally** block syntax −

```
<script type="text/javascript">
   <!--
      try {
         // Code to run
         [break;]
      }

      catch ( e ) {
         // Code to run if an exception occurs
         [break;]
      }

      [ finally {
         // Code that is always executed regardless of
         // an exception occurring
      }]
   //-->
</script>
```

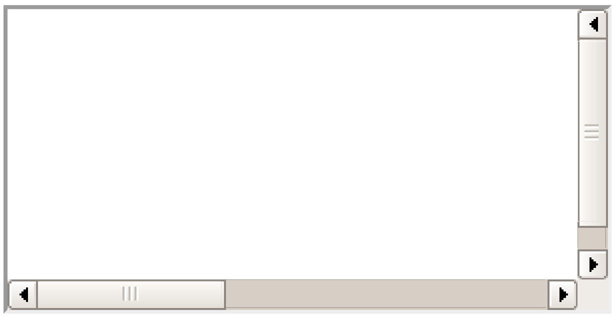
The **try** block must be followed by either exactly one **catch** block or one **finally** block *oroneofboth*. When an exception occurs in the **try** block, the exception is placed in **e** and the **catch** block is executed. The optional **finally** block executes unconditionally after try/catch.

Examples

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**−

```
<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>
```

Output

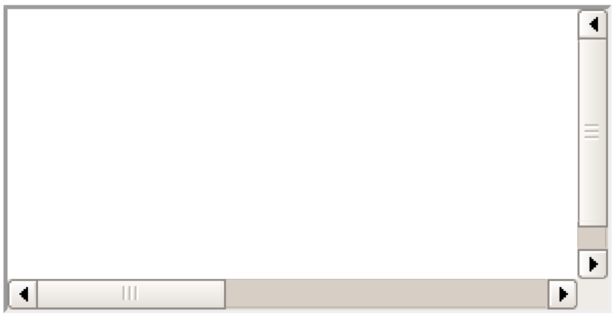


Now let us try to catch this exception using **try...catch** and display a user-friendly message. You can also suppress this message, if you want to hide this error from a user.

```
<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>
```

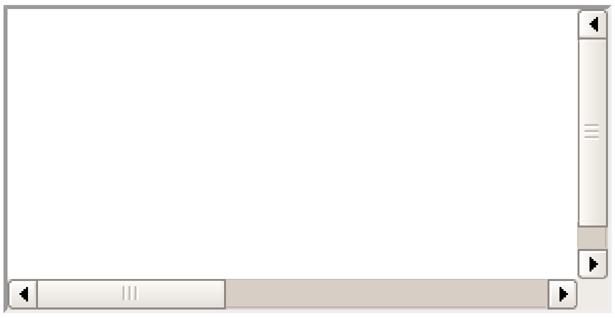
Output



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

```
<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>
```

Output



The throw Statement

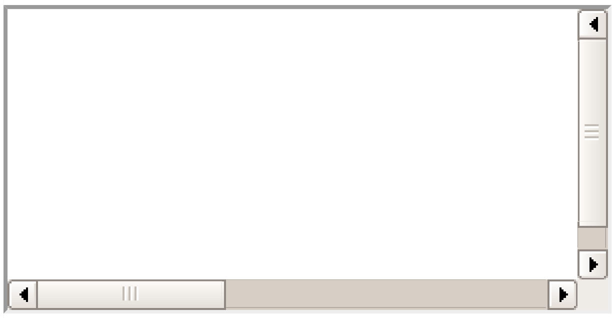
You can use **throw** statement to raise your built-in exceptions or your customized exceptions. Later these exceptions can be captured and you can take an appropriate action.

Example

The following example demonstrates how to use a **throw** statement.

```
<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>
```

Output



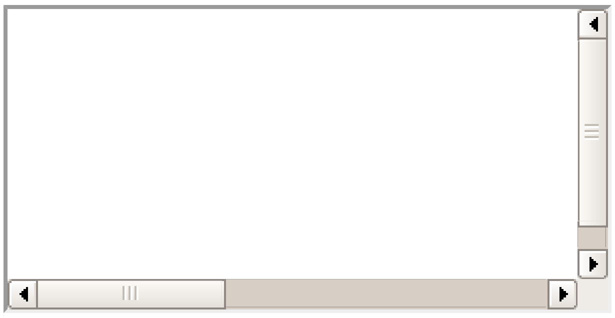
You can raise an exception in one function using a string, integer, Boolean, or an object and then you can capture that exception either in the same function as we did above, or in another function using a **try...catch** block.

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.

```
<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>
```

Output



The **onerror** event handler provides three pieces of information to identify the exact nature of the error –

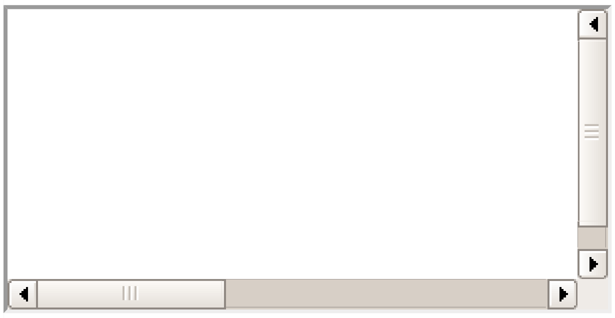
- **Error message** – The same message that the browser would display for the given error
- **URL** – The file in which the error occurred
- **Line number**– The line number in the given URL that caused the error

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

Example

```
<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>
```

Output



You can display extracted information in whatever way you think it is better.

You can use an **onerror** method, as shown below, to display an error message in case there is any problem in loading an image.

```
<img src="myimage.gif" onerror="alert('An error occurred loading the image.')" />
```

You can use **onerror** with many HTML tags to display appropriate messages in case of errors.

Loading [MathJax]/jax/output/HTML-CSS/jax.js