JAVASCRIPT FUNCTION

ITEC87

- A function is a group of reusable code which can be called anywhere in your program.
- This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes.
- Functions allow a programmer to divide a big program into a number of small and manageable functions.

FUNCTION DEFINITION

- Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

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

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

        function sayHello() {
          document.write("Hello there");
        }

</script>

<p>Click the following button to call the function</p>
  <form>
     <input type = "button" onclick = "sayHello()" value = "Say Hello">
  </form>
```

Click the following button to call the function

Say Hello

Hello there

FUNCTION PARAMETERS

- Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

EXAMPLE

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

        function sayHello(name, age) {
          document.write (name + " is " + age + " years old.");
          }

    </script>

    <p>Click the following button to call the function</p>
      <form>
        <input type = "button" onclick = "sayHello('Zara', 7)" value = "Say Hello">
      </form>

</body>
```

Click the following button to call the function

Say Hello

Zara is 7 years old.

RETURN STATEMENT

- A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.
- For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

EXAMPLE

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

   function concatenate(first, last) {
      var full;
      full = first + last;
      return full;
   }

   function secondFunction() {
      var result;
      result = concatenate('Zara', 'Ali');
      document.write (result );
   }

</script>

<p>Click the following button to call the function</p>
  <form>
     <input type = "button" onclick = "secondFunction()" value = "Call Function">
  </form>
```

Click the following button to call the function

Call Function

ZaraAli

JAVASCRIPT ERRORS

- There are three types of errors in programming:
  (a) Syntax Errors
  (b) Runtime Errors
  (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>
```

RUNTIME ERRORS

- Runtime errors, also called exceptions, occur during execution (after compilation/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.

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

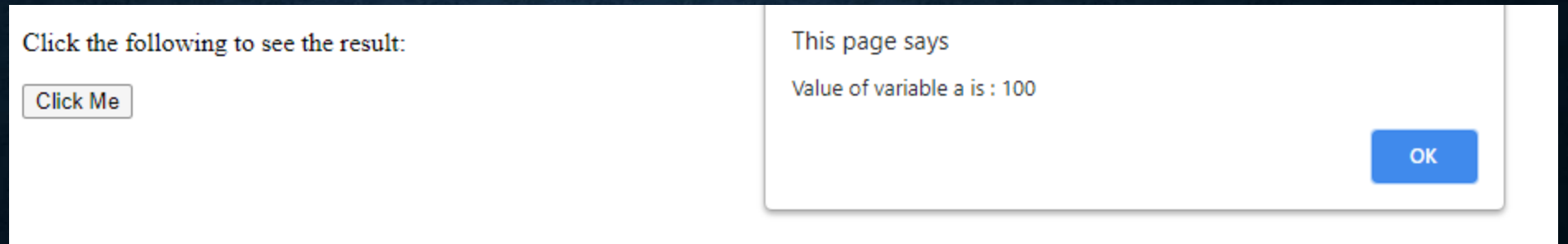
- The latest versions of JavaScript added exception handling capabilities. JavaScript implements the try…catch…finally construct as well as the throw operator to handle exceptions.
- You can catch programmer-generated and runtime exceptions, but you cannot catch JavaScript syntax errors.

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>
```

EXAMPLE

```
<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>
```



```
<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>
```

```
<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>
```

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

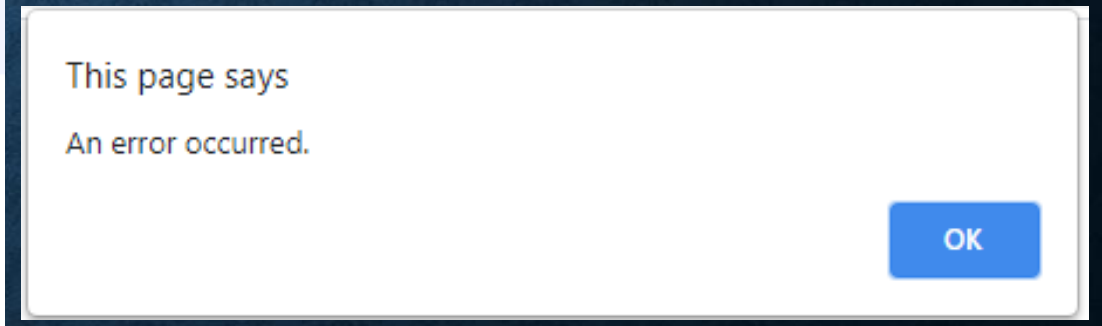
```
<html>
   <head>
      <script type = "text/javascript">
         <!--
            function myFunc() {
               var a = 100;
               var b = 0;

               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>
```

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.

EXAMPLE

```
<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 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

- end