Javascript

It covers basic terms involved in javascript.

Table of Content

How to link Js to html file

The `src` attribute specifies the URL of an external script file.

If you want to run the same JavaScript on several pages in a web site, you should create an external JavaScript file, instead of writing the same script over and over again. Save the script file with a .js extension, and then refer to it using the `src` attribute in the `<script>` tag.

Note: The external script file cannot contain the `<script>` tag.

Note: Point to the external script file exactly where you would have written the script

Browsr that support:

Chrome

Firfox

safari

How to link Js to html file

Syntax

<script src="URL"

The URL of the external script file.

Possible values:

An absolute URL - points to another web site (like src="http://www.example.com/example.js")

A relative URL - points to a file within a web site (like src="/scripts/example.js")

The loggin and the output

Java Logger

In Java, **logging** is an important feature that helps developers to trace out the errors. Java is the programming language that comes with the **logging** approach. It provides a **Logging API** that was introduced in Java 1.4 version. It provides the ability to capture the log file. In this section, we are going to deep dive into the **Java Logger API**. Also, we will cover **logging level, components, Logging handlers or appenders, logging formatters or layouts, Java Logger class.**

In Java**Logging** is an API that provides the ability to trace out the errors of the applications. When an application generates the logging call, the Logger records the event in the LogRecord. After that, it sends to the corresponding handlers or appenders. Before sending it to the console or file, the appenders format that log record by using the formatter or layouts.

Components of Logging

The Java Logging components used by the developers to create logs and passes these logs to the corresponding destination in the proper format. There are the following three core components of the Java logging API:

Loggers

Logging Handlers or Appender

Logging Formatters or Layouts

Loggers

The code used by the client sends the log request to the Logger objects. These logger objects keep track of a log level that is interested in, also rejects the log requests that are below this level. In other words, it is responsible for capturing log records. After that, it passes the records to the corresponding appender.

Generally, the Loggers objects are named entities. The entities separated by the dot operator. For example, java.net, java.awt, etc.

In the logging namespace, the Logger search for the parent loggers. It is closest to the extant ancestor in the logging namespace. Remember that there is no parent for the root logger. It inherits the various attributes from their parent such as:

- **Logging Levels:** If it is specified to null, it traverses towards the parent and search for the first non-null level.
- **Handlers:** a logger will log any message to the handlers of the parent recursively up to the tree.
- **Resource bundle names:** If any logger has a null resource bundle name, it inherits any resource bundle name for its parent recursively up to the tree

Logging Handlers or Appender

Java Logging API allows us to use multiple handlers in a Java logger and the handlers process the logs accordingly. There are the five logging handlers in Java:

- **StreamHandler:** It writes the formatted log message to an OutputStream.
- **ConsoleHandler:** It writes all the formatted log messages to the console.
- **FileHandler:** It writes the log message either to a single file or a group of rotating log files in the XML format.
- **SocketHandler:** It writes the log message to the remote TCP ports.
- **MemoryHandler:** It handles the buffer log records resides in the memory.

In the Java Logging API, the **FileHandler** and **ConsoleHandler** are the two handlers provided by default. By extending the **Handler** class or any subclasses (like MemoryHandler, StreamHandler, etc.), we can create our own and also customize the Handler class. the selection of the appenders depends on the logging requirements. If you are not sure which appenders to use, it may affect the application's performance.

Logging Formatters or Layouts

The logging formatters or layouts are used to format the log messages and convert data into log events. Java SE provides the following two standard formatters class:

- SimpleFormatter
- XMLFormatter

SimpleFormatter

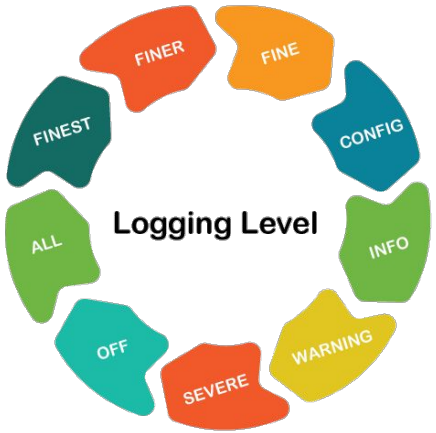
It generates a text message that has general information. It writes human-readable summaries of log messages. The ConsoleHandler uses the class to print the log message to the console.

XMLFormatter

The XMLFormatter generates the log message in XML format. It writes the detailed XML structure information. It is the default formatter for the FileHandler.

Java Logging Levels

It shows the rough guide to the importance and urgency of the log message, and also controls the logging details. Each log level object has an integer value. The higher value indicates higher priorities. There is a total of **nine** levels, **seven** standard logs, and **two** special log levels. The first three logging levels FINEST, FINER, and FINE represent the detailed tracing information that includes what is going on in the application and what happened in the application.



- **FINEST:** It represents the highly detailed tracing message.
- **FINER:** It represents the detailed tracing message that includes exceptions thrown by the application, logging details of the method.
- **FINE:** It represents the most important message out of these.
- **CONFIG:** It represents the information related to the configuration of the application. The information may include how much the disk and memory space.
- **INFO:** The information of the user used by the administrator or other authorities.
- **WARNING:** It occurs due to user mistake. If a user inputs the wrong credentials, the application shows a warning.
- **SEVERE:** It occurs when some critical or terrible errors shown by the application. In such cases, the application is not able to continue further. The popular example of a severe level is out of memory and unavailability of the database.

Logging Frameworks

We can also use the logging framework to make the logging concept easy. There is the following popular logging framework used for logging:

- **Log4j:** Apache Log4j is an open-source Java-based logging utility.
- **SLF4J:** It stands for Simple Logging Facade for Java (SLF4J). It is an abstraction layer for multiple logging frameworks such as Log4j, Logback, and java.util.logging.
- **Logback:** It is an open-source project designed as a successor to Log4j version 1 before Log4j version 2 was released.
- **tinylog(tinylog):** It is a light weighted and open-source logger.

there is more information about bloggers feel free to research later

how to declare a variable

What are Variables?

Variables are containers for storing data (storing data values).

In this example, x, y, and z, are variables, declared with the var keyword:

Example

```
var x = 5;

var y = 6;

var z = x + y;
```

Ways to Declare a JavaScript Variable:

- Using var
- Using let
- Using const
- Using nothing

When to Use JavaScript var?

Always declare JavaScript variables with var,let, orconst.

The var keyword is used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015.

If you want your code to run in older browsers, you must use var.

When to Use JavaScript const?

If you want a general rule: always declare variables with const.

If you think the value of the variable can change, use let.

In this example, price1, price2, and total, are variables:

Example

```
const price1 = 5;

const price2 = 6;

let total = price1 + price2;
```

Data Types

Java Data Types

As explained in the previous chapter, a variable in Java must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99f;    // Floating point number
char myLetter = 'D';         // Character
boolean myBool = true;       // Boolean
String myText = "Hello";     // String
```

Data types are divided into two groups:

- Primitive data types - includes `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` and `char`
- Non-primitive data types - such as `String`, Arrays and Classes (you will learn more about these in a later chapter)

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods

There are eight primitive data types in Java

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
Short	2 bytes	Stores whole numbers from -32,768 to 32,767

nt 4 bytes Stores whole numbers from -2,147,483,648 to 2,147,483,647

Long 8 bytes Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

Float 4 bytes Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits

Double 8 bytes Stores fractional numbers. Sufficient for storing 15 decimal digits

Boolean 1 bit Stores true or false values

Char 2 bytes Stores a single character/letter or ASCII values

string Concatenation

String concatenation means add strings together.

Use the + character to add a variable to another variable:

Example

```
x = "Python is "

y = "awesome"

z =  x + y

print(z)
```

String methods help you to work with strings

String Methods and Properties

Primitive values, like "John Doe", cannot have properties or methods (because they are not objects).

But with JavaScript, methods and properties are also available to primitive values, because JavaScript treats primitive values as objects when executing methods and properties.

Method	Description
charAt()	Returns the character at the specified index.
charCodeAt()	Returns the Unicode of the character at the specified index.
concat()	Joins two or more strings, and returns a new string.
endsWith()	Checks whether a string ends with a specified substring.
fromCharCode()	Converts Unicode values to characters.
includes()	Checks whether a string contains the specified substring.
indexOf()	Returns the index of the first occurrence of the specified value in a string.
lastIndexOf()	Returns the index of the last occurrence of the spe

localeCompare()	Compares two strings in the current locale.
match()	Matches a string against a regular expression, and returns an array of all matches.
repeat()	Returns a new string which contains the specified number of copies of the original string.
replace()	Replaces the occurrences of a string or pattern inside a string with another string, and return a new string without modifying the original string.
search()	Searches a string against a regular expression, and returns the index of the first match.
slice()	Extracts a portion of a string and returns it as a new string.
split()	Splits a string into an array of substrings.
startsWith()	Checks whether a string begins with a specified substring.
substr()	Extracts the part of a string between the start index and a number of characters after it.
substring()	Extracts the part of a string between the start and end indexes.
toLocaleLowerCase()	Converts a string to lowercase letters, according to host machine's current locale.

toLowerCase()	Converts a string to lowercase letters.
toString()	Returns a string representing the specified object.
toUpperCase()	Converts a string to uppercase letters.
trim()	Removes whitespace from both ends of a string.
valueOf()	Returns the primitive value of a String object.

String Properties

The following table lists the standard properties of the String object.

Property	Description
length	Returns the length of a string.
prototype	Allows you to add new properties and methods to an String object.

The JavaScript Array Object

The JavaScript Array object is a global object that is used in the construction of arrays. An array is a special type of variable that allows you to store multiple values in a single variable.

To learn more about Arrays, please check out the JavaScript array chapter.

Array Properties

The following table lists the standard properties of the Array object.

Property	Description
length	Sets or returns the number of elements in an array.
prototype	Allows you to add new properties and methods to an Array object.

Array Methods

The following table lists the standard methods of the Array object.

Method	Description
concat()	Merge two or more arrays, and returns a new array.
copyWithin()	Copies part of an array to another location in the same array and returns it.
entries()	Returns a key/value pair Array Iteration Object.
every()	Checks if every element in an array pass a test in a testing function.
fill()	Fill the elements in an array with a static value.
filter()	Creates a new array with all elements that pass the test in a testing function.
find()	Returns the value of the first element in an array that pass the test in a testing function.
findIndex()	Returns the index of the first element in an array that pass the test in a testing function.

forEach()	Calls a function once for each array element.
from()	Creates an array from an object.
includes()	Determines whether an array includes a certain element.
indexOf()	Search the array for an element and returns its first index.
isArray()	Determines whether the passed value is an array.
join()	Joins all elements of an array into a string.
keys()	Returns a Array Iteration Object, containing the keys of the original array.
lastIndexOf()	Search the array for an element, starting at the end, and returns its last index.
map()	Creates a new array with the results of calling a function for each array element.
pop()	Removes the last element from an array, and returns that element.

reduce()	Reduce the values of an array to a single value (from left-to-right).
reduceRight()	Reduce the values of an array to a single value (from right-to-left).
reverse()	Reverses the order of the elements in an array.
shift()	Removes the first element from an array, and returns that element.
slice()	Selects a part of an array, and returns the new array.
some()	Checks if any of the elements in an array passes the test in a testing function.
sort()	Sorts the elements of an array.
splice()	Adds/Removes elements from an array.
toString()	Converts an array to a string, and returns the result.
unshift()	Adds new elements to the beginning of an array, and returns the array's new length.
values()	Returns a Array Iteration Object, containing the values of the original array.

Different Types of Loops in JavaScript

Loops are used to execute the same block of code again and again, as long as a certain condition is met. The basic idea behind a loop is to automate the repetitive tasks within a program to save the time and effort. JavaScript now supports five different types of loops:

- **while** — loops through a block of code as long as the condition specified evaluates to true.
- **do…while** — loops through a block of code once; then the condition is evaluated. If the condition is true, the statement is repeated as long as the specified condition is true.
- **for** — loops through a block of code until the counter reaches a specified number.
- **for…in** — loops through the properties of an object.
- **for…of** — loops over iterable objects such as arrays, strings, etc.

The while Loop

This is the simplest looping statement provided by JavaScript.

The `while` loop loops through a block of code as long as the specified condition evaluates to true. As soon as the condition fails, the loop is stopped. The generic syntax of the while loop is:

```
while(condition) {
   // Code to be executed
}
```

The following example defines a loop that will continue to run as long as the variable `i` is less than or equal to 5. The variable `i` will increase by 1 each time the loop runs

The do...while Loop

The do-while loop is a variant of the while loop, which evaluates the condition at the end of each loop iteration. With a do-while loop the block of code executed once, and then the condition is evaluated, if the condition is true, the statement is repeated as long as the specified condition evaluated to is true. The generic syntax of the do-while loop is:

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

The JavaScript code in the following example defines a loop that starts with i=1. It will then print the output and increase the value of variable i by 1. After that the condition is evaluated, and the loop will continue to run as long as the variable i is less than, or equal to 5.

Difference Between while and do...while Loop

The `while` loop differs from the `do-while` loop in one important way — with a `while` loop, the condition to be evaluated is tested at the beginning of each loop iteration, so if the conditional expression evaluates to false, the loop will never be executed.

With a `do-while` loop, on the other hand, the loop will always be executed once even if the conditional expression evaluates to false, because unlike the `while` loop, the condition is evaluated at the end of the loop iteration rather than the beginning.

The for Loop

The for loop repeats a block of code as long as a certain condition is met. It is typically used to execute a block of code for certain number of times. Its syntax is:

```
for(initialization; condition; increment) {
   // Code to be executed
}
```

The parameters of the for loop statement have following meanings:

- ***initialization*** — it is used to initialize the counter variables, and evaluated once unconditionally before the first execution of the body of the loop.
- ***condition*** — it is evaluated at the beginning of each iteration. If it evaluates to true, the loop statements execute. If it evaluates to false, the execution of the loop ends.
- ***increment*** — it updates the loop counter with a new value each time the loop runs.

The following example defines a loop that starts with i=1. The loop will continued until the value of variable i is less than or equal to 5. The variable i will increase by 1 each time the loop runs

The for loop is particularly useful for iterating over an array. The following example will show you how to print each item or element of the JavaScript array.

The for...in Loop

The `for-in` loop is a special type of a loop that iterates over the properties of an object, or the elements of an array. The generic syntax of the `for-in` loop is:

```
for(variable in object) {
  // Code to be executed
}
```

The loop counter i.e. *variable* in the `for-in` loop is a string, not a number. It contains the name of current property or the index of the current array element.

The for...of Loop ES6

ES6 introduces a new for-of loop which allows us to iterate over arrays or other iterable objects (e.g. strings) very easily. Also, the code inside the loop is executed for each element of the iterable object.

CONDITIONALS

Conditional statements control behavior in JavaScript and determine whether or not pieces of code can run.

There are multiple different types of conditionals in JavaScript including:

“If” statements: where if a condition is true it is used to specify execution for a block of code.

“Else” statements: where if the same condition is false it specifies the execution for a block of code.

“Else if” statements: this specifies a new test if the first condition is false.

Now that you have the basic JavaScript conditional statement definitions, let’s show you examples of each.

If Statement Example

As the most common type of conditional, the if statement only runs if the condition enclosed in parentheses () is truthy.

EXAMPLE

```
if (10 > 5) {var outcome = "if block";}

outcome;
```

OUTPUT

"if block"

Here's what's happening in the example above:

The keyword if tells JavaScript to start the conditional statement.

(10 > 5) is the condition to test, which in this case is true — 10 is greater than 5.

The part contained inside curly braces {} is the block of code to run.

Because the condition passes, the variable outcome is assigned the value "if block"

Else Statement Example

You can extend an if statement with an else statement, which adds another block to run when the if conditional doesn't pass.

EXAMPLE

```
if ("cat" === "dog") {

    var outcome = "if block";

} else {var outcome = "else block";}

outcome;
```

OUTPUT

"else block"

In the example above, "cat" and "dog" are not equal, so the else block runs and the variable outcome gets the value "else block".

Else If Statement Example

You can also extend an if statement with an else if statement, which adds another conditional with its own block.

EXAMPLE

```
if (false) {
      var outcome = "if block";
} else if (true) {  var outcome = "else if block";
} else { var outcome = "else block";}
outcome;
```

OUTPUT

"else if block"

You can use multiple if else conditionals, but note that only the first else if block runs.

An else if statement doesn't need a following else statement to work. If none of the if or else if conditions pass, then JavaScript moves forward and doesn't run any of the conditional blocks of code.

EXAMPLE

```
if (false) {
    var outcome = "if block";
} else if (false) {var outcome = "else if block";}
outcome;
```

OUTPUT

"first else if block"

JavaScript Logical Operators

The logical operators are typically used to combine conditional statements.

Operator	Name	Example	Result
&&	And	x && y	True if both x and y are true
||	Or	x || y	True if either x or y is true
!	Not	!x	True if x is not true