JavaScript

* runs on the client side of the web
* can be used to design / program how the web pages behave on the occurrence of an event
* dynamic scripting language supporting prototype based object construction
* can function as both a procedural and an object oriented language.
* JavaScript's dynamic capabilities include runtime object construction, variable parameter lists, function variables
* cross-platform
* contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements.
  + *Client-side JavaScript* extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
  + *Server-side JavaScript* extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.
* Firefox console for single line code, Scratch Pad for multi lines
* **case-sensitive**
* use undefined (primitive type) to determine whether a variable has a value
  + undefined value converts to NaN when used in numeric context
* runs on the client side of the web

**Data types**

The latest ECMAScript standard defines seven data types:

* Six data types that are [primitives](https://developer.mozilla.org/en-US/docs/Glossary/Primitive):
  + [Boolean](https://developer.mozilla.org/en-US/docs/Glossary/Boolean). true and false.
  + [null](https://developer.mozilla.org/en-US/docs/Glossary/null). A special keyword denoting a null value. Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant.
  + [undefined](https://developer.mozilla.org/en-US/docs/Glossary/undefined). A top-level property whose value is undefined.
  + [Number](https://developer.mozilla.org/en-US/docs/Glossary/Number). 42 or 3.14159.
  + [String](https://developer.mozilla.org/en-US/docs/Glossary/String). "Howdy"
  + [Symbol](https://developer.mozilla.org/en-US/docs/Glossary/Symbol) (new in ECMAScript 2015). A data type whose instances are unique and immutable.
* and [Object](https://developer.mozilla.org/en-US/docs/Glossary/Object)
* in expressions involving numeric and string values with the + operator, JavaScript converts numeric values to strings.
* does not convert numeric values to strings. For example:
  + "37" - 7 // 30
  + "37" + 7 // "377"
* In the case that a value representing a number is in memory as a string, there are methods for conversion.
  + [parseInt()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt)
  + [parseFloat()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat)
* literals
* You use literals to represent values in JavaScript. These are fixed values, not variables, that you literally provide in your script. This section describes the following types of literals:
* [Array literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Array_literals)
* [Boolean literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Boolean_literals)
* [Floating-point literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Floating-point_literals)
* [Integers](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Integers)
* [Object literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#Object_literals)
* [RegExp literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#RegExp_literals)
* [String literals](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_types#String_literals)
* Primitive parameters (such as a number) are passed to functions **by value**
* If you pass an object (i.e. a non-primitive value, such as [Array](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function
* function can be **anonymous**; it does not have to have a name

To summarize:

* The inner function can be accessed only from statements in the outer function.
* The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.
* Since the inner function forms a closure, you can call the outer function and specify arguments for both the outer and inner function:
* When two arguments or variables in the scopes of a closure have the same name, there is a name conflict. More inner scopes take precedence, so the inner-most scope takes the highest precedence, while the outer-most scope takes the lowest.
* Closures are one of the most powerful features of JavaScript. JavaScript allows for the nesting of functions and grants the inner function full access to all the variables and functions defined inside the outer function (and all other variables and functions that the outer function has access to). However, the outer function does not have access to the variables and functions defined inside the inner function. This provides a sort of security for the variables of the inner function.
* Using the arguments object, you can call a function with more arguments than it is formally declared to accept. This is often useful if you don't know in advance how many arguments will be passed to the function. You can use arguments.length to determine the number of arguments actually passed to the function, and then access each argument using the arguments object.
* In JavaScript, parameters of functions default to undefined. However, in some situations it might be useful to set a different default value. This is where default parameters can help.
* The [rest parameter](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/rest_parameters) syntax allows us to represent an indefinite number of arguments as an array.
* == return true if equal
* ===equal+sametype
* The [delete](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/delete) operator deletes an object, an object's property, or an element at a specified index in an array
* <https://msdn.microsoft.com/en-us/library/aa479011.aspx>