typeof

The **`typeof`** operator returns a string indicating the type of the unevaluated operand.

JavaScript Demo: Expressions - typeof

```
console.log(typeof 42);
// expected output: "number"

console.log(typeof 'blubber');
// expected output: "string"

console.log(typeof true);
// expected output: "boolean"

console.log(typeof declaredButUndefinedVariable);
// expected output: "undefined";

```

Run ›

Reset

```
> "number"
> "string"
> "boolean"
> "undefined"
```

Syntax

The `typeof` operator is followed by its operand:

```
typeof operand
typeof(operand)
```

Parameters

`operand`

An expression representing the object or primitive whose type is to be returned.

Description

The following table summarizes the possible return values of `typeof`. For more information about types and primitives, see also the JavaScript data structure page.

Type	Result
Undefined	`"undefined"`
Null	`"object"` (see below)
Boolean	`"boolean"`
Number	`"number"`
BigInt	`"bigint"`
String	`"string"`

Type	Result
Symbol (new in ECMAScript 2015)	`"symbol"`
Host object (provided by the JS environment)	*Implementation-dependent*
Function object (implements [[Call]] in ECMA-262 terms)	`"function"`
Any other object	`"object"`

Examples

```
// Numbers
typeof 37 === 'number';
typeof 3.14 === 'number';
typeof(42) === 'number';
typeof Math.LN2 === 'number';
typeof Infinity === 'number';
typeof NaN === 'number'; // Despite being "Not-A-Number"
typeof Number('1') === 'number'; // Number tries to parse things into numbers

typeof 42n === 'bigint';


// Strings
```

```
typeof '' === 'string';
typeof 'bla' === 'string';
typeof `template literal` === 'string';
typeof '1' === 'string'; // note that a number within a string is still typeof string
typeof (typeof 1) === 'string'; // typeof always returns a string
typeof String(1) === 'string'; // String converts anything into a string, safer than toStrin


// Booleans
typeof true === 'boolean';
typeof false === 'boolean';
typeof Boolean(1) === 'boolean'; // Boolean() will convert values based on if they're truthy
typeof !!(1) === 'boolean'; // two calls of the ! (logical NOT) operator are equivalent to B


// Symbols
typeof Symbol() === 'symbol'
typeof Symbol('foo') === 'symbol'
typeof Symbol.iterator === 'symbol'


// Undefined
typeof undefined === 'undefined';
typeof declaredButUndefinedVariable === 'undefined';
typeof undeclaredVariable === 'undefined';


```

```
// Objects
typeof {a: 1} === 'object';

// use Array.isArray or Object.prototype.toString.call
// to differentiate regular objects from arrays
typeof [1, 2, 4] === 'object';

typeof new Date() === 'object';
typeof /regex/ === 'object'; // See Regular expressions section for historical results


// The following are confusing, dangerous, and wasteful. Avoid them.
typeof new Boolean(true) === 'object';
typeof new Number(1) === 'object';
typeof new String('abc') === 'object';


// Functions
typeof function() {} === 'function';
typeof class C {} === 'function';
typeof Math.sin === 'function';
```

Additional information

null

```
// This stands since the beginning of JavaScript
typeof null === 'object';
```

In the first implementation of JavaScript, JavaScript values were represented as a type tag and a value. The type tag for objects was 0. `null` was represented as the NULL pointer (0x00 in most platforms). Consequently, null had 0 as type tag, hence the "object" `typeof` return value. (reference)

A fix was proposed for ECMAScript (via an opt-in), but was rejected. It would have resulted in `typeof null === 'null'`.

Using `new` operator

```
// All constructor functions, with the exception of the Function constructor, will always be
var str = new String('String');
var num = new Number(100);

typeof str; // It will return 'object'
typeof num; // It will return 'object'

var func = new Function();

```

```
typeof func; // It will return 'function'
```

Need for parentheses in Syntax

```
// Parentheses can be used for determining the data type of expressions.
var iData = 99;

typeof iData + ' Wisen'; // 'number Wisen'
typeof (iData + ' Wisen'); // 'string'
```

Regular expressions

Callable regular expressions were a non-standard addition in some browsers.

```
typeof /s/ === 'function'; // Chrome 1-12 Non-conform to ECMAScript 5.1
typeof /s/ === 'object';   // Firefox 5+  Conform to ECMAScript 5.1
```

Errors

Before ECMAScript 2015, `typeof` was always guaranteed to return a string for any operand it was supplied with. Even with undeclared identifiers, `typeof` will return `'undefined'`. Using `typeof` could never generate an error.

But with the addition of block-scoped `let` and `Statements/const` using `typeof` on `let` and `const` variables (or using `typeof` on a `class`) in a block before they are declared will throw a `ReferenceError`. Block scoped variables are in a "temporal dead zone" from the start of the block until the initialization is processed, during which, it will throw an error if accessed.

```
typeof undeclaredVariable === 'undefined';

typeof newLetVariable; // ReferenceError
typeof newConstVariable; // ReferenceError
typeof newClass; // ReferenceError

let newLetVariable;
const newConstVariable = 'hello';
class newClass{};
```

Exceptions

All current browsers expose a non-standard host object `document.all` with type `undefined`.

```
typeof document.all === 'undefined';
```

Although the specification allows custom type tags for non-standard exotic objects, it requires those type tags to be different from the predefined ones. The case of `document.all` having type `'undefined'` is classified in the web standards as a "willful violation" of the original ECMA JavaScript standard.

Specifications

Specification	Status	Comment
ECMAScript Latest Draft (ECMA-262) The definition of 'The typeof Operator' in that specification.	D Draft	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'The typeof Operator' in that specification.	ST Standard	
ECMAScript 5.1 (ECMA-262) The definition of 'The typeof Operator' in that specification.	ST Standard	
ECMAScript 3rd Edition (ECMA-262) The definition of 'The typeof Operator' in that specification.	ST Standard	
ECMAScript 1st Edition (ECMA-262) The definition of 'The typeof Operator' in that specification.	ST Standard	Initial definition. Implemented in JavaScript 1.1.

Browser compatibility

Update compatibility data on GitHub

`typeof`

Chrome	Yes
Edge	Yes
Firefox	1
IE	Yes
Opera	Yes
Safari	Yes
WebView Android	Yes
Chrome Android	Yes
Edge Mobile	Yes
Firefox Android	4
Opera Android	Yes
Safari iOS	Yes
Samsung Internet Android	Yes
nodejs	Yes

Full support

IE-specific notes

On IE 6, 7, and 8 a lot of host objects are objects and not functions. For example:

```
typeof alert === 'object'
```

See also

- `instanceof`
- Why typeof is no longer "safe"
- document.all willful violation of the standard