Arguments object

The **`arguments`** object is an `Array`-like object corresponding to the arguments passed to a function.

Syntax

```
arguments
```

Description

The `arguments` object is a local variable available within all functions. `arguments` as a property of `Function` can no longer be used.

You can refer to a function's arguments within the function by using the `arguments` object. This object contains an entry for each argument passed to the function, the first entry's index starting at 0. For example, if a function is passed three arguments, you can refer to the argument as follows:

```
arguments[0]
arguments[1]
arguments[2]
```

The arguments can also be set:

```
arguments[1] = 'new value';
```

The `arguments` object is not an `Array`. It is similar to an `Array`, but does not have any `Array` properties except `length`. For example, it does not have the `pop` method. However it can be converted to a real `Array`:

```
var args = Array.prototype.slice.call(arguments);
```

Important: You should not slice on arguments because it prevents optimizations in JavaScript engines (V8 for example). Instead, try constructing a new array by iterating through the arguments object. More information.

The `arguments` object is available only within a function body. Attempting to access the `arguments` object outside a function declaration results in an error.

You can use the `arguments` object if you call a function with more arguments than it is formally declared to accept. This technique is useful for functions that can be passed a variable number of arguments. You can use `arguments.length` to determine the number of arguments passed to the function, and then process each argument by using the `arguments` object. (To determine the number of arguments declared when a function was defined, use the `Function.length` property.)

Properties

`arguments.callee`
Reference to the currently executing function.

`arguments.caller`
Reference to the function that invoked the currently executing function.

`arguments.length`
Reference to the number of arguments passed to the function.

Examples

Defining a function that concatenates several strings

This example defines a function that concatenates several strings. The only formal argument for the function is a string that specifies the characters that separate the items to concatenate. The function is defined as follows:

```
function myConcat(separator) {
  var args = Array.prototype.slice.call(arguments, 1);
  return args.join(separator);
}
```

You can pass any number of arguments to this function, and it creates a list using each argument as an item in the list.

```
// returns "red, orange, blue"
myConcat(", ", "red", "orange", "blue");

// returns "elephant; giraffe; lion; cheetah"
myConcat("; ", "elephant", "giraffe", "lion", "cheetah");

// returns "sage. basil. oregano. pepper. parsley"
myConcat(". ", "sage", "basil", "oregano", "pepper", "parsley");
```

Defining a function that creates HTML lists

This example defines a function that creates a string containing HTML for a list. The only formal argument for the function is a string that is "`u`" if the list is to be unordered (bulleted), or "`o`" if the list is to be ordered (numbered). The function is defined as follows:

```
function list(type) {
  var result = "<" + type + "l><li>";
  var args = Array.prototype.slice.call(arguments, 1);
  result += args.join("</li><li>");
  result += "</li></" + type + "l>"; // end list

  return result;
}
```

You can pass any number of arguments to this function, and it adds each argument as an item to a list of the type indicated. For example:

```
var listHTML = list("u", "One", "Two", "Three");

/* listHTML is:

"<ul><li>One</li><li>Two</li><li>Three</li></ul>"

*/
```

Rest, default and destructured parameters

The `arguments` object can be used in conjunction with rest parameters, default parameters or destructured parameters.

```
function foo(...args) {
  return arguments;
}
foo(1, 2, 3); // { "0": 1, "1": 2, "2": 3 }
```

However, in non-strict functions, a **mapped `arguments` object** is only provided if the function does **not** contain any rest parameters, any default parameters or any destructured parameters. For example, in the following function that uses a default parameter, 10 instead of 100 is returned:

```
function bar(a=1) {
  arguments[0] = 100;
  return a;
}
```

```
bar(10); // 10
```

Specifications

Specification	Status	Comment
ECMAScript 1st Edition (ECMA-262)	ST Standard	Initial definition. Implemented in JavaScript 1.1
ECMAScript 5.1 (ECMA-262) The definition of 'Arguments Object' in that specification.	ST Standard	
ECMAScript 2015 (6th Edition, ECMA-262) The definition of 'Arguments Exotic Objects' in that specification.	ST Standard	
ECMAScript 2016 Draft (7th Edition, ECMA-262) The definition of 'Arguments Exotic Objects' in that specification.	D Draft	

Browser compatibility

Desktop Mobile

Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari
Basic support	(Yes)	(Yes)	(Yes)	(Yes)	(Yes)

See also

- `Function`