Functions are**first-class objects** and they provide **scope**.

• Can be created dynamically at runtime, during the execution of the program

• Can be assigned to variables, can have their references copied to other variables, can be augmented, and, except for a few special cases, can be deleted

• Can be passed as arguments to other functions and can also be returned by other functions

• Can have their own properties and methods

[复制代码](javascript:void(0);)

// antipattern

// for demo purposes only

var add = new Function('a, b', 'return a + b');

add(1, 2); // returns 3

[复制代码](javascript:void(0);)

Any variable defined with  **var** inside of a function is a local variable, invisible outside the function. Saying that curly braces don’t provide local scope means that if you define a variable with  **var** inside of an  **if** condition or inside of a  **for** or a **while** loop, that doesn’t mean the variable is local to that i for for.

**Variable scope**

It’s only local to the wrapping function, and if there’s no wrapping function, it becomes a global variable.

**Disambiguation of Terminology**

function expression/ anonymous function

[复制代码](javascript:void(0);)

// function expression, a.k.a. anonymous function

var add = function (a, b) {

return a + b;

};

[复制代码](javascript:void(0);)

named function expression

[复制代码](javascript:void(0);)

// named function expression

var add = function **add**(a, b) {

return a + b;

};

[复制代码](javascript:void(0);)

**Note**

The only difference is that the name property of the function object will be a blank string. The name property is an extension of the language (it’s not part of the ECMA standard) but widely available in many environments.

The name property is useful when using debuggers, such as Firebug, or when calling the same function recursively from itself.

**function declarations**

function foo() {

// function body goes here

}

**Declarations Versus Expressions: Names and Hoisting**

[复制代码](javascript:void(0);)

// this is a function expression, passed as an argument to the function `callMe`

callMe(function () {

// I am an unnamed function expression

// also known as an anonymous function

});

// this is a named function expression

callMe(function me() {

// I am a named function expression

// and my name is "me"

});

// another function expression

var myobject = {

say: function () {

// I am a function expression

}

};

[复制代码](javascript:void(0);)

Function  declarations can only appear in “program code,” meaning inside of the bodies of other functions or in the global space. Their definitions cannot be assigned to variables or properties, or appear in function invocations as parameters.

[复制代码](javascript:void(0);)

// global scope

function foo() {}

function local() {

// local scope

function bar() {}

return bar;

}

[复制代码](javascript:void(0);)

**Function’s name Property**

The availability of the read-only  name property.

[复制代码](javascript:void(0);)

function foo() {} // declaration

var bar = function () {}; // expression

var baz = function baz() {}; // named expression

foo.name; // "foo"

bar.name; // "" // undefined in IE; empty string in FF, Webkit

baz.name; // "baz"

[复制代码](javascript:void(0);)

The case against function declarations and the reason to prefer **function expressions** is that the expressions **highlight that functions are objects** like all other objects and not some special language construct.

**Note**

Don't assign a different name to a named function expression since it's not properly implemented in some browsers(IE).

var foo = function bar() {};

**Function Hoisting**

[复制代码](javascript:void(0);)

// antipattern

// for illustration only

// global functions

function foo() {

alert('global foo');

}

function bar() {

alert('global bar');

}

function hoistMe() {

console.log(typeof foo); // "function"

console.log(typeof bar); // "undefined"

foo(); // "local foo"

bar(); // TypeError: bar is not a function

// function declaration:

// variable 'foo' and its implementation both get hoisted

function foo() {

alert('local foo');

}

// function expression:

// only variable 'bar' gets hoisted

// not the implementation

var bar = function () {

alert('local bar');

};

}

hoistMe();

[复制代码](javascript:void(0);)

Just like with normal variables, the mere presence of foo and  bar anywhere in the hoistMe() function moves them to the top, overwriting the global foo and bar. The difference is that local  foo()’s  definition is hoisted to the top and works fine; although it’s defined later. The definition of bar()is not hoisted, only its  declaration.  That’s  why  until  the  code  execution  reaches  bar()’s  definition,  it’s undefined and not usable as a function (while still preventing the global  bar()from being “seen” in the scope chain).