Node.js Data Type

Node.js is highly performant, and it uses JavaScript because JavaScript supports first-class functions and closures.

Node.js has a few core types: number , boolean , string , and object.

The value undefined means that a value has not been set yet or simply does not exist:

**var** x;

console.log(x);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/D/DESCRIPTION__FC4C0F3560BEEAC35642.PNG

Null

null is an explicit assertion that there "is no value":

**var** y;

console.log(y);

y = null ;

console.log(y);

The code above generates the following result.



typeof

To see the type of anything in JavaScript, use the typeof operator:

console.log(typeof 10);

console.log(typeof **"hello"**);

console.log(typeof function () { **var** x = 20; });

The code above generates the following result.



Constants

The standard practice is to use uppercase letters and variable declarations:

**var** SECONDS\_PER\_DAY = 86400;

console.log(SECONDS\_PER\_DAY);

Type Comparisons and Conversions

JavaScript has both the equality operator == and the precise equality operator ===.

console.log(234 == **'234'**);

console.log(234 === **'234'**);

console.log(234234.235235 == **'cat'**);

console.log(**"cat"** == **"CAT"**);

console.log(**"cat"**.toUpperCase() == **"CAT"**);

The code above generates the following result.



A number of different values evaluate to false.

console.log(**''** == false == null == undefined == 0);

console.log(null === undefined);

To check arguments to functions:

function fine(param) {

if (param == null || param == undefined || param == **''**)

throw new Error(**"Invalid Argument"**);

}

function better(param) {

if (!param)

throw new Error(**"Invalid Argument"**);

}

Pay more attention to the primitive wrapper.

**var** x = 234;

**var** x1 = new Number(234);

console.log(typeof x);

console.log(typeof x);

console.log(x1 == x);

console.log(x1 === x);

The code above generates the following result.

# **Node.js Numbers**

All numbers in JavaScript are 64-bit IEEE 754 double-precision floating-point numbers.

All numbers in JavaScript have the same floating point number type.

Arithmetic operations (+,-,\*,/,%) work on numbers as you would expect.

**var** myData = 1;

**var** myValue = 2;

console.log(myData + 1);

console.log(myData / myValue);

console.log(myData \* myValue);

console.log(myData - myValue);

console.log(myData % 2);

The code above generates the following result.

Example

The number type in JavaScript behaves much like integer data types in other languages:

console.log(1024 \* 1024);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__52C24A011A203613DA15.PNG

The tricky part of using the number type, however, is that for many numeric values, it is an approximation of the actual number.

For example:

console.log(0.1 + 0.2);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__8877ECE09DB03341F431.PNG

When performing floating-point mathematical operations, we cannot manipulate arbitrary real numbers and expect an exact value:

console.log(1 - 0.3 + 0.1 == 0.8);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__9B61B41A00346212B41F.PNG

Note

In JavaScript dividing a number by zero returns the value Infinity or -Infinity instead of generating a runtime exception:

console.log(5 / 0);

console.log(-5 / 0);

The code above generates the following result.



Infinity and -Infinity are valid values that we can compare against in JavaScript:

**var** x = 10, y = 0;

console.log(x / y == Infinity);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/N/NOTE__E79CB79318B529DD69B9.PNG

Note 2

You can use the functions parseInt and parseFloat to convert strings to numbers:

console.log(parseInt(**"32"**));

console.log(parseFloat(**"8.24"**));

console.log(parseInt(**"234.12345"**));

console.log(parseFloat(**"10"**));

The code above generates the following result.



Note 3

If we provide these functions with not-parsable value, they return the special value NaN:

console.log(parseInt(**"cat"**));

console.log(parseFloat(**"css"**));

The code above generates the following result.



To test for NaN, use the isNaN function:

isNaN(parseInt(**"cat"**));

Note 4

To test whether a given number is a valid finite number, use the isFinite function:

console.log(isFinite(10/5));

console.log(isFinite(10/0));

console.log(isFinite(parseFloat(**"css"**)));

The code above generates the following result.



# **Node.js Booleans**

The boolean type Values in JavaScript can either be true or false.

Two literals are defined for boolean values: true and false.

We can convert values to boolean with the Boolean function, and the language converts everything to boolean when needed, according to the following rules:

false, 0 , empty strings "", NaN , null , and undefined all evaluate to false .

All other values evaluate to true .

## Example

console.log(0 == false);/\*from ww w. j a va 2 s .com\*/

console.log(**""** == false);

if(null){

}else{

console.log(**"false"**);

}

if(undefined){

}else{

console.log(**"false"**);

}

if(NaN){

}else{

console.log(**"false"**);

}

The code above generates the following result.

Example 2

We can assign these to variables and apply boolean operations to them.

**var** myData = true;

console.log(myData); **// true**

/\*from w w w . java 2s . co m\*/

**// Boolean operations (&&, ||, !) work as expected:**

console.log(true && true); **// true**

console.log(true && false); **// false**

console.log(true || false); **// true**

console.log(false || false); **// false**

console.log(!true); **// false**

console.log(!false); **// true**

The code above generates the following result.



# **Node.js Strings**

Strings in JavaScript are sequences of Unicode characters.

We use a string of length 1 to represent character.

Strings can be wrapped in single or double quotation marks.

They are functionally equivalent.

To include a single quotation mark inside a single-quoted string, we can use \', and similarly for double quotation marks inside double-quoted strings, we can use \":

console.log(**'Javascript\'s new feature.'**)

console.log(**"\"Hey, new feature!\", he said."**)

The code above generates the following result.



Example

To get the length of a string in JavaScript, just use the length property:

**var** x = **"cat"**;

console.log(x.length);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__C3232778B557BD0C8BFE.PNG

To add two strings together, you can use the + operator:

**var** s = **"CSS"** + **" is "** + **"cool"**;

console.log(s);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__0FD94C4F24F236C0385D.PNG

Javascript + can convert them as best it can:

**var** distance = 25;

**var** s = **"This is a test: "** + distance + **"."**;

console.log(s);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__35928671F74BA8CC3AF2.PNG

indexOf

To find a string with another string, use the indexOf function:

**var** i = **"this is a test"**.indexOf(**"is"**);

console.log(i);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/I/INDEXOF__B241A13069718E460AB1.PNG

substr and splice

To extract a substring from a string, use the substr or splice function.

substr takes the starting index and length of string to extract. splice takes the starting index and ending index:

**var** s = **"this is a test string."**.substr(19, 3);

**var** s1 = **"this is a test string."**.slice(19, 22);

console.log(s);

console.log(s1);

The code above generates the following result.



## Split

To split string into substrings, use the split function and get an array as the result:

**var** s = **"a|b|c|d|e|f|g|h"**.split(**"|"**);

console.log(s);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/S/SPLIT__07796568995FFDBD2133.PNG

The trim function from V8 Javascript function removes whitespace from the beginning and end of a string:

var s = ' cat \n\n\n '. trim();

console.log(s);

# **Node.js Arrays**

To create arrays, you can either use traditional notation or array literal syntax :

var arr1 = new Array();

var arr2 = [];

As with objects, the literal syntax version is preferred.

We can test if an object is an array using the Array.isArray function:

var arr2 = [];

Array.isArray(arr2);

Array.isArray({});

We can create arrays quite easily in JavaScript using [].

Arrays have many useful functions.

**var** myData = [];

// w ww. j a va 2 s. c o m

myData.push(1); **// add at the end**

console.log(myData); **// prints [1]**

myData.unshift(2); **// add to the top**

console.log(myData); **// prints [2,1]**

**// Arrays are zero index based:**

console.log(myData[0]); **// prints 2**

The code above generates the following result.

Example

The array type in JavaScript length property returns the element count.

**var** arr2 = [];

arr2.length;

console.log(arr2);

**var** arr3 = [ **'cat'**, **'rat'**, **'bat'** ];

console.log(arr3.length);

The code above generates the following result.



By default, arrays in JavaScript are numerically indexed:

**var** arr3 = [ **'cat'**, **'rat'**, **'bat'** ];

for (**var** i = 0; i < arr3.length; i++) {

console.log(arr3[i]);

}

The code above generates the following result.



To add an item to the end of an array, you can do one of two things:

**var** arr3 = [ **'cat'**, **'rat'**, **'bat'** ];

arr3.push(**"mat"**);

console.log(arr3);

arr3[arr3.length] = **"fat"**;

console.log(arr3);

The code above generates the following result.



We can specify the index of the element where you want to insert a new element.

If this element is past the last element, the elements in between are created and initialized with the value undefined :

**var** arr3 = [ **'cat'**, **'rat'**, **'bat'** ];

arr3[20] = **"splat"**;

console.log(arr3);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__C3A0ECA0EF949B532F59.PNG

## Note

To delete an item from an array, use the splice function, which takes an index and the number of items to delete.

What it returns is an array with the extracted items, and the original array is modified such that they no longer exist there:

**var** arr3 = [ **'cat'**, **'rat'**, **'bat'**, **'cat'**, **'rat'**, **'bat'** ];

arr3.splice(2, 2);

console.log(arr3);

console.log(arr3.length);

The code above generates the following result.

# **Node.js Array Functions**

push and pop

The push and pop functions let you add and remove items to the end of an array, respectively:

**var** nums = [ 1, 1, 2, 3, 5, 8 ];

nums.push(13);

console.log(nums);

nums.pop();

console.log(nums);

The code above generates the following result.



unshift and shift

To insert or delete items from the front of an array, use unshift or shift , respectively:

**var** nums = [ 1, 2, 3, 5, 8 ];

nums.unshift(1);

console.log(nums);

nums.shift();

console.log(nums);

join

The array function join returns a string from the array:

**var** nums = [ 1, 1, 2, 3, 5, 8 ];

**var** s = nums.join(**", "**);

console.log(s);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/J/JOIN__FF5D8886697560C77EA2.PNG

Sort

You can sort arrays using the sort function, which can be used with the built-in sorting function:

**var** nums = [ 3, 1, 8, 5, 2, 1];

nums.sort();

console.log(nums);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/S/SORT__409EE71626E0C3D004F5.PNG

We can provide your own sorting function as a parameter:

**var** names = [ **'CSS'**, **'HTML'**, **'Java'**, **'SQL'**, **'CSS3'**, **'HTML5'**];

names.sort();/\* w w w . j a va 2 s . co m\*/

console.log(names);

names.sort(function (a, b) {

**var** a1 = a.toLowerCase(), b1 = b.toLowerCase();

if (a1 < b1) return 1;

if (a1 > b1) return -1;

return 0;

});

console.log(names);

The code above generates the following result.

## http://www.java2s.com/Tutorials/JavascriptImage/myResult/S/SORT__6809103DE90DC1F03990.PNGLoop

To iterate over items in arrays, we can use the for loop or forEach function

[ **'CSS'**, **'HTML'**, **'CSS3'**, **'HTML5'**, **'Javascript'**, **'SQL'**].forEach( function (value) {

console.log(value);

});

The code above generates the following result.

# **Node.js Functions**

JavaScript is a functional programming language, functions are fully typed objects that can be manipulated, extended, and passed around as data.

A normal function structure in JavaScript is defined as follows.

function functionName() {

// function body

// optional return;

}

All functions return a value in JavaScript.

In the absence of an explicit return statement, a function returns undefined.

function myData() {

return 123;

}

console.log(myData()); **// 123**

function myValue() {

}

console.log(myValue()); **// undefined**

Example

The following code shows how to create a function:

function hello(name) {

console.log(**"hello "** + name);

}

hello(**"CSS"**);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE__C1290A0563F49A0A4A98.PNG

Note

To declare parameters for a function in JavaScript, list them in the parentheses.

There is no checking of these parameters at runtime:

function hello(name) {

console.log(**"hello "** + name);

}

hello();

hello(**"CSS"**, **"HTML"**, **"AAA"**, 4);

The code above generates the following result.



If too few parameters are passed into a function call, the resulting variables are assigned the value undefined.

If too many are passed in, the extras are simply unused.

All functions have a predefined array in the body called arguments.

It has all the values that were passed in to the function, and we can do extra checking on the parameter list.

Functions in JavaScript do not even need to have names:

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

return a + b;

}

console.log(x(10, 20));

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/N/NOTE__5BA2E967DAE9E3B6274A.PNG

The nameless functions are typically called anonymous functions.

## Function Scope

Every time a function is called, a new variable scope is created.

Variables declared in the parent scope are available to that function.

Variables declared within the new scope are not available when the function exits.

Consider the following code:

**var** pet = **'cat'**;

function myMethod() {

**var** pet = **'dog'**;

console.log(pet);

}

myMethod();

console.log(pet);

The code above generates the following result.



Combining this scoping with anonymous functions is better way to use private variables that will disappear when the anonymous function exits.

Example 2

Here's a contrived example to compute the volume of a cone:

**var** **height** = 5;

**var** radius = 3;

**var** volume;

**// declare and immediately call anonymous function to create scope**

(function () {/\*from w w w . j av a 2 s . c o m\*/

**var** pir2 = Math.PI \* radius \* radius; **// temp var**

volume = (pir2 \* **height**) / 3;

})();

console.log(volume);

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/E/EXAMPLE_2__2BBA1C1E0F133FE3058F.PNG

# **Node.js Higher Order Function**

## Immediately Executing Function

We can execute a function immediately after you define it.

Simply wrap the function in parentheses () and invoke it.

(function myData() {

console.log(**'myData was executed!'**);

})();

An immediately executing function creates a new variable scope.

An if, else, or while does not create a new variable scope in JavaScript.

**var** myData = 123;

if (true) {

**var** myData = 456;

}

console.log(myData); **// 456;**

The only recommended way of creating a new variable scope in JavaScript is using a function.

The following code shows how to create a new variable scope with an immediately executing function.

**var** myData = 123;

if (true) {

(function () { **// create a new scope**

**var** myData = 456;

})();

}

console.log(myData); **// 123;**

## Anonymous Function

A function without a name is called an anonymous function.

In JavaScript, you can assign a function to a variable.

If you are going to use a function as a variable, you don't need to name the function.

The following code shows two ways of defining a function inline.

Both of these methods are equivalent.

**var** foo1 = function namedFunction() {

console.log(**'foo1'**);

}

foo1(); **// foo1**

**var** foo2 = function () { **// no function name i.e. anonymous function**

console.log(**'foo2'**);

}

foo2(); **// foo2**

A programming language is said to have first-class functions if a function can be treated the same way as any other variable in the language.

JavaScript has first-class functions.

## Higher-Order Functions

Since JavaScript allows us to assign functions to variables, we can pass functions to other functions.

Functions that take functions as arguments are called higher-order functions.

A very common example of a higher-order function is setTimeout.

The following code shows how to use setTimeout function.

setTimeout(function () {

console.log(**'2000 milliseconds have passed since this demo started'**);

}, 2000);

If you run this application in Node.js, you will see the console.log message after two seconds and then the application will exit.

We used an anonymous function as the first argument to setTimeout.

This makes setTimeout a higher-order function.

We can create a function and passing that in.

function foo() {

console.log(**'2000 milliseconds have passed since this demo started'**);

}

setTimeout(foo, 2000);

## Closures

If there is a function defined inside another function, the inner function has access to the variables declared in the outer function.

The variables in the outer function have been closed by the inner function.

The concept in itself is simple enough and fairly intuitive.

function outerFunction(arg) {

**var** variableInOuterFunction = arg;

function myValue() {

console.log(variableInOuterFunction);

}

myValue();

}

outerFunction(**'hello closure!'**); **// logs hello closure!**

# **Node.js Objects**

To create an object, we can use either of the following,

**var** o1 = new Object();

**var** o2 = {};

The latter, known as object literal syntax, is preferred.

We can specify the contents of objects using object literal syntax.

We can specify member names and values at initialization time:

## Example

**var** user = {

first\_name: **"HTML"**,

last\_name: **"CSS"**,

age: 32,

website: **"java2s.com"**

};

We can add a new property to your user object by using any of the following methods:

user.name= **"brown"**;

user[**"name"**] = **"brown"**;

**var** attribute = **'name'**;

user[attribute] = **"brown"**;

If we try to access a property that does not exist, we do not receive an error, but instead just get back undefined.

To remove a property from an object, we can use the delete keyword:

delete user.name;

Object Literals

The most common way of creating an object in JavaScript is using the object notation, {}.

Objects can be extended arbitrarily at runtime.

**var** myData = {};

console.log(myData); **// {}**

myData.myValue = 123; **// extend myData**

console.log(myData); **// { myValue: 123 }**

The code above generates the following result.



We can define which properties go on an object upfront by using the object literal notation.

**var** myData = {

myValue: 123

};

console.log(myData); **// { myValue: 123 }**

The code above generates the following result.

http://www.java2s.com/Tutorials/JavascriptImage/myResult/O/OBJECT_LITERALS__BA213F0C50B026309795.PNG

We can nest object literals inside object literals.

**var** myData = {

myValue: 123,

bas: {

bas1: **'some string'**,

bas2: 345

}

};

console.log(myData);

We can have arrays inside object literals.

**var** myData = {

myValue: 123,

bas: [1, 2, 3]

};

console.log(myData);

We can also have these arrays themselves contain object literals.

**var** myData = {

myValue: 123, /\*from w ww .j a v a 2 s . co m\*/

bas: [{

myItem: 1

},

{

myItem: 2

},

{

myItem: 3

}]

};

console.log(myData.myValue); **// 123**

console.log(myData.bas[0].myItem); **// 1**

console.log(myData.bas[2].myItem); **// 2**