Day 0: Data Types

Data Types

JavaScript's Data Types

The latest ECMAScript standard defines seven data types:

- A *primitive* value or data type is data that is not an object and has no methods. All primitives are immutable, meaning they cannot be changed. There are six primitive types:
  - *Number*
  - *String*
  - *Boolean*
  - *Symbol*
  - *Null*
  - *Undefined*
- The seventh data type is *Object*

Number Data Type

According to the ECMAScript standard, all numbers are double-precision 64-bit binary format IEEE 754-2008, meaning there is no specific type for integers.

Maximum Value for a Number

The `MAX_VALUE` property has a value of approximately $1.7976931348623157 \times 10^{308}$. Values larger than `Number.MAX_VALUE` are represented as `Infinity`.

Minimum Value for a Number

The `MIN_VALUE` property is the smallest positive value of the *Number* type closest to 0, not the most negative number, that JavaScript can represent. `MIN_VALUE` has a value of approximately 5×10^{-324}. Values smaller than `Number.MIN_VALUE` ("underflow values") are converted to 0.

Symbolic Numbers

There are three symbolic number values:

- `Infinity`: This is any number divided by 0, or an attempt to multiply `Number.MAX_VALUE` by an integer > 1.
- `-Infinity`: This is any number divided by -0, or an attempt to multiply `Number.MAX_VALUE` by an integer < -1.

Go to Top

- `NaN`: This stands for "Not-a-Number" and denotes an unrepresentable value (i.e., $\sqrt{-1}$).

The *isSafeInteger* Method

A *safe integer* is an integer that:

- Can be exactly represented as an IEEE-754 double precision number, and
- Whose IEEE-754 representation cannot be the result of rounding any other integer to fit the IEEE-754 representation.

The `Number.isSafeInteger()` method determines whether the provided value is a number that is a safe integer.

Maximum Safe Integer

The `Number.MAX_SAFE_INTEGER` constant has a value of 9007199254740991, or $2^{53} - 1$.

Minimum Safe Integer

The `Number.MIN_SAFE_INTEGER` constant has a value of -9007199254740991, or $-2^{53} + 1$.

- EXAMPLE

Run the code below to learn more about the Number type.

```
var var1 = 1;
var var2 = 0;
var var3 = -0;

console.log("1 / 0 = " + var1 / var2);
console.log("1 / -0 = " + var1 / var3);
console.log("MAX_VALUE: " + Number.MAX_VALUE);
console.log("MAX_VALUE + 1 = " + Number.MAX_VALUE * 2);
console.log("MIN_VALUE = " + Number.MIN_VALUE);
console.log("MIN_VALUE - 1 = " + Number.MIN_VALUE - 1);
console.log("MAX_SAFE_INTEGER = " + Number.MAX_SAFE_INTEGER);
console.log("MIN_SAFE_INTEGER = " + Number.MIN_SAFE_INTEGER);
console.log("SquareRoot(-1) = " + Math.sqrt(-1));
```

Output

Run

String Data Type

A string value is a chain of zero or more Unicode characters (i.e., letters, digits, and punctuation marks) that we use to represent text. We include string literals in our scripts by enclosing them in single (`'`) or double (`"`) quotation marks. Double quotation marks can be contained in strings surrounded by single quotation marks (e.g., `'"'` evaluates to `"`), and single quotation marks can be contained in strings surrounded by double quotation marks (e.g., `"'"` evaluates to `'`). The following are examples of strings:

- EXAMPLE

Run the code below to learn more about the String type.

```
var firstString = "Hello, There.";
var secondString = "How're you?";
var thirdString = "c";
var fourthString = '"Wow!!!", she shouted.';

console.log(firstString);
console.log(secondString);
```

Table Of Contents

Go to Top

```
8 console.log(thirdString);
```

Output

Run

Notice that JavaScript does not have a type to represent a single character. To represent a single character in JavaScript, you create a string that consists of only one character. A string that contains zero characters ("") is an empty (zero-length) string.

Unlike in languages like C, JavaScript strings are immutable. This means that once a string is created, it is not possible to modify it. However, it is still possible to create another string based on an operation on the original string. For example:

- A substring of the original by picking individual letters or using `String.substr()`.
- A concatenation of two strings using the concatenation operator (`+`) or `String.concat()`.

Boolean Data Type

A boolean represents a logical entity and can have one of two literal values: `true`, and `false`.

Symbol Data Type

Symbols are new to JavaScript in ECMAScript Edition 6. A Symbol is a unique and immutable primitive value and may be used as the key of an Object property.

Null Data Type

The null data type is an internal type that has only one value: `null`. This is a primitive value that represents the absence of any object value. A variable that contains null contains no valid number, string, boolean, array, or object. You can erase the contents of a variable (without deleting the variable) by assigning it the null value.

Undefined Data Type

The undefined value is returned when you use an object property that does not exist, or a variable that has been declared, but has never had a value assigned to it.

The *typeof* Operator

As demonstrated in some of the code examples above, we can use the `typeof` operator to determine the type associated with a variable's current value:

\- EXAMPLE

Run the code below to learn more about `typeof`.

```
// Number Data Type:
var firstVar = 1.5e5;

// String Data Type:
var secondVar = 'Hello';

// Boolean Data Type:
var thirdVar = true;
```

Table Of Contents

Go to Top

```
// Symbol Data Type:
var fourthVar = Symbol("some Symbol variable");

// Null Object:
var fifthVar = null;

// Undefined Data Type:
var sixthVar;

// Object:
var seventhVar = {a: 1, b: 2};

// NaN (It is a Number):
var eighthVar = Math.sqrt(-1);

console.log(firstVar + " is a " + typeof firstVar);
console.log(secondVar + " is a " + typeof secondVar);
console.log(thirdVar + " is a " + typeof thirdVar);
console.log(fourthVar.toString() + " is a " + typeof fourthVar);
console.log(fifthVar + " is an " + typeof fifthVar);
console.log(sixthVar + " is an " + typeof sixthVar);
console.log(seventhVar + " is an " + typeof seventhVar);
```

Output

Run

Variables

Dynamic Typing

JavaScript is a loosely typed or *dynamic* language, meaning you don't need to declare a variable's type ahead of time and the language autmatically determines a variable's type while the program is being processed. That also means that you can reassign a single variable to reference different types. For example:

\- EXAMPLE

Run the code below to learn more about dynamic typing.

```
function print() {
    console.log(
        "someVariable(" + someVariable
        + ") is a " + typeof someVariable
    );
    // Note: 'typeof' is explained later in this tutorial.
}

// Declare someVariable and initialize it to the number '5':
var someVariable = 5;
print(someVariable);

// Assign the string "Hello" to someVariable:
var someVariable = "Hello";
print(someVariable);

// Assign the boolean value 'true' to someVariable:
var someVariable = true;
```

Output

Run

Naming

Go to Top

JavaScript is a case-sensitive language, meaning that a variable name such as `myVariable` is different from the variable name `myvariable`. Variable names can be of any length, and the rules for creating legal variable names are as follows:

- The first character must be either an ASCII letter (uppercase or lowercase) or an underscore (`_`). Note that a number *cannot* be used as the first character.
- Subsequent characters can be ASCII letters, underscores, or digits (e.g., the numbers **0** through **9**).
- The variable name must not be a reserved word.

The code below declares some *valid* variable names:

```
var _daysCount
var MinimumCost
var page10
var Total_elements
```

The following declarations are *invalid* variable names and will not compile:

```
// This will produce "SyntaxError: Unexpected number"
var 10Students

// This will produce "SyntaxError: Unexpected token &"
var First&Second
```

Declaration and Initialization

The first time a variable appears in your script is considered its *declaration*. The first mention of the variable sets it up in memory, and the name allows you to refer back to it in your subsequent lines of code. You should declare variables using the `var` keyword before using them. If you do not initialize a variable that was declared using the `var` keyword, it automatically takes on the value `undefined`.

\- EXAMPLE

Consider the following code:

```
// Declare firstVar:
var firstVar;

// Initialize firstVar:
firstVar = 1;

// Declare and Initialize secondVar:
var secondVar = "String";

// Declare thirdVar and fourthVar:
var thirdVar,
    fourthVar;

// Initialize thirdVar:
thirdVar = true;

// Initialize fourthVar:
fourthVar = null;

// Declare and Initialize fifthVar and sixthVar:
var fifthVar = 1.01,
    sixthVar = "Sixth";

// Declare seventhVar:
var seventhVar;

console.log(firstVar);
console.log(secondVar);
console.log(thirdVar);
```

Table Of Contents

Go to Top

```
30 console.log(fourthVar);
31 console.log(fifthVar);
32 console.log(sixthVar);
```

Output

Run

Coercion

In JavaScript, you can perform operations on values of different types without raising an exception. The JavaScript interpreter implicitly converts, or coerces, one of the data types to that of the other, then performs the operation. The rules for coercion of string, number, or boolean values are as follows:

- If you add a number and a string, the number is coerced to a string.
- If you add a boolean and a string, the boolean is coerced to a string.
- If you add a number and a boolean, the boolean is coerced to a number.

\- EXAMPLE

Run the code below to learn more about type coercion.

```
1  function print(name, variable) {
2      console.log(
3          name + "(" + variable
4          + ") is a " + typeof variable
5      );
6  }
7
8  var someNumber = 10;
9  var someString = "Ten";
10 var someBoolean = false;
11
12 var sumOfNumberAndString = someNumber + someString;
13 var sumOfBooleanAndString = someBoolean + someString;
14 var sumOfNumberAndBoolean = someNumber + someBoolean;
15
16 print("sumOfNumberAndString", sumOfNumberAndString);
17 print("sumOfBooleanAndString", sumOfBooleanAndString);
```

Output

Run

Table Of Contents