UNIVERSIDADE da MADEIRA

CTeSP CURSOS TÉCNICOS SUPERIORES PROFISSIONAIS

Tecnologias e Programação de Sistemas de Informação

JAVASCRIPT

Desenvolvimento Web - Back-End | David Jardim

Cofinanciado por:

MADEIRA 14-20

REGIÃO AUTÓNOMA DA MADEIRA

PORTUGAL 2020

UNIÃO EUROPEIA
Fundo Social Europeu

DATA-TYPES in JAVASCRIPT

- PRIMITIVES
- NON-PRIMITIVES
- DYNAMIC OR *“LOOSELY-TYPED”*

PRIMITIVE DATA TYPES

- *String*
- *Number*
- *Boolean*
- *Null*
- *Undefined*

```
var name = "Peter";     // String
var counter = 5;        // Number
var hasEnded = false;   // Boolean
var value = null;       // null
var undef;              // undefined
```

NON-PRIMITIVE DATA TYPES

- *Array*
- *Date*
- *Object*

```
var cars = ["Saab","Volvo","BMW"];
var date = new Date();
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

STRING

- The String type is the set of all finite ordered sequences of zero or more 16-bit unsigned integer values (“elements”)

- Each element is considered to be a single UTF-16 code unit

```
var str1 = "Hello World";
```

- Concatenation:

```
var str = 'Hello ' + "World " + 'from ' + 'TutorialsTeacher ';
```

- String as array:

```
var str = 'Hello World';

str[0] // H
str[1] // e
str[2] // l
str[3] // l
str[4] // o
```

Method	Description
charAt(position)	Returns the character at the specified position (in Number).
charCodeAt(position)	Returns a number indicating the Unicode value of the character at the given position (in Number).
concat([string,,])	Joins specified string literal values (specify multiple strings separated by comma) and returns a new string.
indexOf(SearchString, Position)	Returns the index of first occurrence of specified String starting from specified number index. Returns -1 if not found.
lastIndexOf(SearchString, Position)	Returns the last occurrence index of specified SearchString, starting from specified position. Returns -1 if not found.
localeCompare(string,position)	Compares two strings in the current locale.
match(RegExp)	Search a string for a match using specified regular expression. Returns a matching array.
replace(searchValue, replaceValue)	Search specified string value and replace with specified replace Value string and return new string. Regular expression can also be used as searchValue.
search(RegExp)	Search for a match based on specified regular expression.
slice(startNumber, endNumber)	Extracts a section of a string based on specified starting and ending index and returns a new string.
split(separatorString, limitNumber)	Splits a String into an array of strings by separating the string into substrings based on specified separator. Regular expression can also be used as separator.
substr(start, length)	Returns the characters in a string from specified starting position through the specified number of characters (length).
substring(start, end)	Returns the characters in a string between start and end indexes.
toLocaleLowerCase()	Converts a string to lower case according to current locale.
toLocaleUpperCase()	Converts a sting to upper case according to current locale.
toLowerCase()	Returns lower case string value.
toString()	Returns the value of String object.
toUpperCase()	Returns upper case string value.
valueOf()	Returns the primitive value of the specified string object.

NUMBER

- JavaScript Number can store various numeric values like integer, float, hexadecimal, decimal, exponential and octal values

```
var int = 100;
var float = 100.5;
var hex = 0xfff;
var exponential = 2.56e3;
var octal = 030;
```

- Number type includes default properties - MAX_VALUE, MIN_VALUE, NEGATIVE_INFINITY, NaN and POSITIVE_INFINITY.

- Number methods are used to perform different tasks on numbers.

Method	Description
toExponential(fractionDigits)	Returns exponential value as a string. **Example:** `var num = 100; Num.toExponential(2); // returns '1.00e+2'`
toFixed(fractionDigits)	Returns string of decimal value of a number based on specified fractionDigits. **Example:** `var num = 100; Num.toFixed(2); // returns '100.00'`
toLocaleString()	Returns a number as a string value according to a browser's locale settings. **Example:** `var num = 100; Num.toLocaleString(); // returns '100'`
toPrecision(precisionNumber)	Returns number as a string with specified total digits. **Example:** `var num = 100; Num.toPrecision(4); // returns '100.0'`
toString()	Returns the string representation of the number value. **Example:** `var num = 100; Num.toString(); // returns '100'`
valueOf()	Returns the value of Number object. **Example:** `var num = new Number(100); Num.valueOf(); // returns '100'`

BOOLEAN

- JavaScript Boolean data type can store one of two values, **true** or **false**.
- JavaScript treats an empty string (""), 0, *undefined* and *null* as false. Everything else is true.
- Boolean methods are used to perform different tasks on Boolean values.

```
var YES = true;

var NO = false;
```

```
alert(1 > 2); // false

alert(10< 9); // false

alert(5 == 5); // true
```

NULL

- The Null type has exactly one value, called null
- Null means absence of value

```
var myVar = null;

if (myVar)
    alert("myVar is not null');
else
    alert("myVar is null" );
```

```
var saveButton = document.getElementById("save");

if (saveButton !== null)
        saveButton.submit();
```

UNDEFINED

- The Undefined type has exactly one value, called undefined.
- Any variable that has not been assigned a value has the value undefined.

```
function Sum(val1, val2)
{
    var result = val1 + val2;
}

var result = Sum(5, 5);
alert(result);// undefined
```

```
function Sum(val1, val2)
{
    return val1 + val2;  // val2 is undefined
}

Sum(5);
```

ARRAY

- An array is a special type of variable, which can store multiple values using special syntax
- Every value is associated with numeric index starting with 0

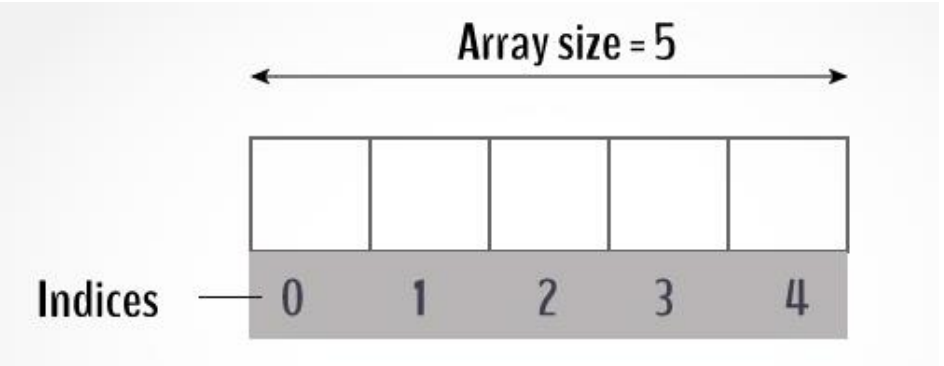
```
var stringArray = ["one", "two", "three"];

var numericArray = [1, 2, 3, 4];

var decimalArray = [1.1, 1.2, 1.3];

var booleanArray = [true, false, false, true];

var mixedArray = [1, "two", "three", 4];
```



Method	Description
concat()	Returns new array by combining values of an array that is specified as parameter with existing array values.
every()	Returns true or false if every element in the specified array satisfies a condition specified in the callback function. Returns false even if single element does not satisfy the condition.
filter()	Returns a new array with all the elements that satisfy a condition specified in the callback function.
forEach()	Executes a callback function for each elements of an array.
indexOf()	Returns the index of the first occurrence of the specified element in the array, or -1 if it is not found.
join()	Returns string of all the elements separated by the specified separator
lastIndexOf()	Returns the index of the last occurrence of the specified element in the array, or -1 if it is not found.
map()	Creates a new array with the results of calling a provided function on every element in this array.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements at the end of an array and returns the new length of the array.
reduce()	Pass two elements simultaneously in the callback function (till it reaches the last element) and returns a single value.
reduceRight()	Pass two elements simultaneously in the callback function from right-to-left (till it reaches the last element) and returns a single value.
reverse()	Reverses the elements of an array. Element at last index will be first and element at 0 index will be last.
shift()	Removes the first element from an array and returns that element.
slice()	Returns a new array with specified start to end elements.
some()	Returns true if at least one element in this array satisfies the condition in the callback function.
sort()	Sorts the elements of an array.
splice()	Adds and/or removes elements from an array.
toString()	Returns a string representing the array and its elements.
unshift()	Adds one or more elements to the front of an array and returns the new length of the array.

DATE

- Get current date using Date() or new Date()
- Date object can be created using new keyword. e.g. var date = new Date();
- Date can be created by specifying milliseconds, date string or year and month in Date constructor.
- Date can be created by specifying date string in different formats using different separators.

```
var dt = new Date();

var dt = new Date(milliseconds);

var dt = new Date('date string');

var dt = new Date(year, month[, date, hour, minute, second, millisecond]);
```

OBJECT LITERAL

- The object literal is a simple way of creating an object using { } brackets
- You can include key-value pair in { }, where key would be property or method name and value will be value of property of any data type or a function
- Use comma (,) to separate multiple key-value pairs

OBJECT LITERAL

```
var emptyObject = {}; // object with no properties or methods

var person = { firstName: "John" }; // object with single property
```

```
// object with properties & method
var person = {
                firstName: "James",
                lastName: "Bond",
                age: 15,
                getFullName: function () {
                        return this.firstName + ' ' + this.lastName
                    }
            };
```

OBJECT LITERAL

```
person.firstName; // returns James
person.lastName; // returns Bond

person["firstName"];// returns James
person["lastName"];// returns Bond

person.getFullName();
```

OBJECT CONSTRUCTOR

- The second way to create an object is with Object Constructor using **new** keyword
- It is possible to attach properties and methods using dot notation
- Optionally, you can also create properties using [ ] brackets and specifying property name as string

```
var person = new Object();

// Attach properties and methods to person object
person.firstName = "James";
person["lastName"] = "Bond";
person.age = 25;
person.getFullName = function () {
        return this.firstName + ' ' + this.lastName;
    };
```

```
// access properties & methods
person.firstName; // James
person.lastName; // Bond
person.getFullName(); // James Bond
```

typeof

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

```
typeof 37 === 'number';
typeof 'bla' === 'string';
typeof false === 'boolean';
typeof declaredButUndefinedVariable === 'undefined';
typeof {a: 1} === 'object';
typeof [1, 2, 4] === 'object';
typeof new Date() === 'object';
```

Operators

- JavaScript includes operators that perform some operation on single or multiple operands (data value) and produce a result
  - Arithmetic Operators
  - Comparison Operators
  - Logical Operators
  - Assignment Operators
  - Conditional Operators

Arithmetic Operators

```
var x = 5, y = 10, z = 15;

x + y; //returns 15

y - x; //returns 5
```

Operator	Description
+	Adds two numeric operands.
-	Subtract right operand from left operand
*	Multiply two numeric operands.
/	Divide left operand by right operand.
%	Modulus operator. Returns remainder of two operands.
++	Increment operator. Increase operand value by one.
--	Decrement operator. Decrease value by one.

Comparison Operators

```
var a = 5, b = 10, c = "5";
var x = a;

a == c; // returns true

a === c; // returns false
```

Operators	Description
==	Compares the equality of two operands without considering type.
===	Compares equality of two operands with type.
!=	Compares inequality of two operands.
>	Checks whether left side value is greater than right side value. If yes then returns true otherwise false.
<	Checks whether left operand is less than right operand. If yes then returns true otherwise false.
>=	Checks whether left operand is greater than or equal to right operand. If yes then returns true otherwise false.
<=	Checks whether left operand is less than or equal to right operand. If yes then returns true otherwise false.

Logical Operators

```
var a = 5, b = 10;

(a != b) && (a < b); // returns true

(a > b) || (a == b); // returns false
```

Operator	Description
&&	&& is known as AND operator. It checks whether two operands are non-zero (0, false, undefined, null or "" are considered as zero), if yes then returns 1 otherwise 0.
||	|| is known as OR operator. It checks whether any one of the two operands is non-zero (0, false, undefined, null or "" is considered as zero).
!	! is known as NOT operator. It reverses the boolean result of the operand (or condition)

Assignment Operators

```
var x = 5, y = 10, z = 15;

x = y; //x would be 10

x += 1; //x would be 6

x -= 1; //x would be 4
```

Assignment operators	Description
=	Assigns right operand value to left operand.
+=	Sums up left and right operand values and assign the result to the left operand.
-=	Subtract right operand value from left operand value and assign the result to the left operand.
*=	Multiply left and right operand values and assign the result to the left operand.
/=	Divide left operand value by right operand value and assign the result to the left operand.
%=	Get the modulus of left operand divide by right operand and assign resulted modulus to the left operand.

Conditional Operators

- JavaScript includes special operator called ternary operator : **?** that assigns a value to a variable based on some condition

- This is like a short form of if-else condition

```
var a = 10, b = 5;

var c = a > b? a : b; // value of c would be 10
var d = a > b? b : a; // value of d would be 5
```

Conditional Statements

- Used to perform different actions based on different conditions

- Can be used in multiple ways:

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Loop For

- Used to perform repetitions or iterate over a data structure
- Alternative to while:

```
var i = 0;
while (i != 10) {
    …
    i++;
}
```

```
for (var i = 0; i != 10; i++) {
    …
}
```


UNIVERSIDADE da MADEIRA
CTeSP
CURSOS TÉCNICOS
SUPERIORES PROFISSIONAIS