# Part 4

## String Manipulation

* JavaScript, unlike most languages, doesn’t have a char type. Instead, characters are just one length strings.
* They are immutable – this means when you change a string you aren’t mutating the string; you’re creating a new one each time.
* Double or single quotes work the same way
  + **‘ ‘** is recommended by most style guides
  + **“ ”** or **` `** is recommended for working with nested  **‘ ‘**
* **String(value)** converts numbers into a string

The String type provides a bunch of useful methods to manipulate string. Information on these methods can be found here:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String>

They take otherwise long-winded processes and condense them into a single method call. Below are some examples of commonly used methods:

const a = "dog".repeat(3);

// a = "dogdogdog"

String.repeat(num) creates a string that’s the original string repeated an input amount of times

const b = "Animals: ".concat(["dog", "cat"]);

// b = "Animals: dog,cat"

String.concat(stringArr) combines a base string with items in an input string array

// String comparison is easy!

"hello".startsWith("ello", 1); //true

"hello".endsWith("hell", 1); //false

"hello".includes("ello"); //true

"hello".includes("ello", 1); //true

"hello".includes("ello", 2); //false

There are a number of comparator methods that check if a string contains a substring

const dog = { age: 10, name: "bob" };

// Trying to assemble a string like this?

console.log("Age: " + dog.age + " Name: " + dog.name);

// This is a much more concise way using ` `

console.log(`Age: ${dog.age} Name: ${dog.name}`);

Using ` ` to assemble a string makes things a lot cleaner

### Escape Character

JavaScript uses **\** as an escape character. Using this we can insert special characters into our strings. Here’s what JavaScript supports:

* \’ – single quote
* \” – double quote
* \\ - backslash
* \n – new line
* \r – carriage return
* \t – tab
* \b - backspace
* \f – form feed
* \v – vertical tab
* \0 – null character

## Booleans

Boolean(value) returns true if the value is truthy, false if the value is falsy.

Below is a list of falsy values.

* false
* null
* undefined
* “” (Empty quotes)
* 0 (zero)
* NaN

**Everything else is truthy.**

## Arrays

* Arrays inherit from object and are **not** a primitive value (like in Java)
* Indexes are converted to strings and used as names for retrieving values
* Very efficient for sparse arrays, not efficient in every other case.
* No need to provide a length or type when creating an array as they are dynamic in size.
* Arrays have a length member. It is always one integer larger than the highest integer subscript.
* [] is preferred to new Array() when creating a new array.

Just like String, Array has its own methods:

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#Methods_2>

Here’s a few common examples:

myArray.splice(1, 1);

Array.splice(startIndex, deleteCount) removes a range of items from an array

myArray.join(', ');

Array.join(separator) creates a string with every element of an array using an input separator

myArray.push('this gets appended');

You can append new items to an array using Array.push(item)

myArray.pop();

Array.pop() removes the last item from an array and returns it

* **Arrays are just objects, special objects!**
* When addressing objects, we can retrieve properties with the array notation as well as the dot notation.
  + **Obj.name**
  + **Obj[name]**
* When you’re addressing an array, all you’re doing is addressing an object with an attribute, the attribute being a number.
* Array[0] is looking for an attribute with 0 as the key

## Objects

* Collections of unordered name/value pairs
* Names are strings
* Values can be any type, including other objects
* Think of every object like a little database.
* Values can also be expressions
* **:** separates names and values
* **,** separates pairs
* Object literals can be used anywhere a value can appear.

//object literal

const myObject = {

name: "Jack",

grade: "A",

level: 3

};

myObject.name = "Jeff";

console.log(myObject.level); //outputs 3

In this example we’re defining an object, updating the value of one of the properties can outputting the value of another

//dynamic object

const exampleObject = new Object();

exampleObject.name = "Jeff";

exampleObject.output = new function () { }

Here we’re using the Object constructor to initialise an empty object. Using **{ }** is also valid.

### Maker Functions

Another way to construct your objects is using a maker function. A maker function is just a function that returns an object using data from parameters.

function maker(name, grade, level) {

let student = {};

student.name = name;

student.grade = grade;

student.level = level;

return student;

}

const myObject = maker("Jack", "A", 3);

That’s a fairly verbose way of doing things. What we could do is this:

function maker(name, grade, level) {

return {

name,

grade,

level

};

}

const myObject = maker("Jack", "A", 3);

This will create the same object as the previous one. If you don’t specify a property name the name of the variable is used. That’s fine for this example, but if we wanted to name our property something different, we can declare it like this **studentName: name**.

### Nested Literals

Objects within objects!

const myObject = {

name: 'Jack',

grade: 'A',

pet: {

type: 'Dog',

width: 19,

height: 10,

weight: 24,

stuff: {}

}

};

You can nest objects just like any other type

## JSON

* JSON stands for **J**avaScript **O**bject **N**otation
* JSON is just syntax for storing and exchanging data
* Data exchanged between a browser and server can only be text.
* JSON itself is text. We can convert any JavaScript object into JSON to send to a sever and convert any received JSON from a server into JavaScript objects.

{

"name": "John",

"gender": "Male",

"age": 37,

"group": [

1,

2,

3,

4

],

"object": {

"name": "Mike",

"age": 29

}

}

This JSON object can be converted and used as a JavaScript object

{

"name": "arrayObjectExample",

"arrayObject": [

{

"name": "Lucy",

"age": 22

},

{

"name": "Ben",

"age": 24

}

]

}

An example of an array of objects

### Parsing JSON

* **JSON.parse()** is used to convert text into a JavaScript Object.
* In the example, using **JSON.parse(text)** would convert the string stored in text into a JavaScript object.
* We can now access the values in the object as shown in the second example which would print John in the paragraph tag.
* **JSON.stringify()** works in the opposite way; it converts a JavaScript object into a JSON formatted string.

const text = `{

"employees" : [

{ "fName":"John" , "sName":"Doe" },

{ "fName":"Anna" , "sName":"Smith" },

{ "fName":"Peter" , "sName":"Jones" }

]

}`;

const obj = JSON.parse(text);

document.getElementById("demo").innerHTML =

obj.employees[1].fName;

<p id="demo"></p>