JavaScript ES6 Cheat Sheet

@codingtute

Arrow function

```
const sum = (a, b) => a + b
console.log(sum(2, 6)) // prints 8
```

Default parameters

```
function print(a = 5) {
 console.log(a)
}
print() // prints 5
print(22) // prints 22
```

Let Scope

```
let a = 3
if (true) {
 let a = 5
 console.log(a) // prints 5
}
console.log(a) // prints 3
```

Const

```
// can be assigned only once
const a = 55
a = 44 // throws an error
```

Multiline string

```
console.log(`
 This is a
 multiline string
`)
```

Template strings

```
const name = 'World'
const message = `Hello ${name}`
console.log(message)
// prints "Hello World"
```

Exponent operator

```
const byte = 2 ** 8
// Same as: Math.pow(2, 8)
```

Spread operator

```
const a = [ 1, 2 ]
const b = [ 3, 4 ]
const c = [ ...a, ...b ]
console.log(c) // [1, 2, 3, 4]
```

String includes()

```
console.log('apple'.includes('pl'))
// prints true
console.log('apple'.includes('tt'))
// prints false
```

String startsWith()

```
console.log('apple'.startsWith('ap'))
//prints true
console.log('apple'.startsWith('bb'))
//prints false
```

String repeat()

```
console.log('ab'.repeat(3))
//prints "abababab"
```

Destructuring array

```
let [a, b] = [3, 7];
console.log(a); // 3
console.log(b); // 7
```

Destructuring object

```
let obj = {
 a: 55,
 b: 44
};
let { a, b } = obj;
console.log(a); // 55
console.log(b); // 44
```

Object property assignment

```
const a = 2
const b = 5
const obj = { a, b }
// Before es6:
// obj = { a: a, b: b }
console.log(obj)
// prints { a: 2, b: 5 }
```

Object.assign()

```
const obj1 = { a: 1 }
const obj2 = { b: 2 }
const obj3 = Object.assign({},
    obj1, obj2)
console.log(obj3)
// { a: 1, b: 2 }
```

Promises with finally

```
promise
 .then((result) => { ··· })
 .catch((error) => { ··· })
 .finally(() => { /* logic
independent of success/error */ })
/* The handler is called when the
promise is fulfilled or rejected.*/
```

Object function assignment

```
const obj = {
 a: 5,
 b() {
 console.log('b')
 }
}
obj.b() // prints "b"
```

Object.entries()

```
const obj = {
 firstName: 'FirstName',
 lastName: 'LastName',
 age: 24,
 country: 'India',
};
const entries = Object.entries(obj);
/* returns an array of [key, value]
 pairs of the object passed */
console.log(entries);
/* prints
 [
['firstName', 'FirstName'],
['lastName', 'LastName'],
['age', 24],
['country', 'India']
]; */
```

Spread operator

```
const a = {
 firstName: "FirstName",
 lastName: "LastName1",
}
const b = {
 ...a,
 lastName: "LastName2",
 canSing: true,
}
console.log(a)
//{firstName: "FirstName", lastName: "LastName1"}
console.log(b)
/* {firstName: "FirstName", lastName: "LastName2",
canSing: true} */
/* great for modifying objects without side
effects/affecting the original */
```

Destructuring Nested Objects

```
const Person = {
 name: "Harry Potter",
 age: 29,
 sex: "male",
 materialStatus: "single",
 address: {
 country: "USA",
 state: "Nevada",
 city: "Carson City",
 pinCode: "500014",
 },
};
const { address : { state, pinCode }, name } = Person;
console.log(name, state, pinCode)
// Harry Potter Nevada 500014
console.log(city) // ReferenceError
```

JavaScript ES6 Cheat Sheet

@codingtute

Arrow function

```
const sum = (a, b) => a + b
console.log(sum(2, 6)) // prints 8
```

Default parameters

```
function print(a = 5) {
 console.log(a)
}
print() // prints 5
print(22) // prints 22
```

Let Scope

```
let a = 3
if (true) {
 let a = 5
 console.log(a) // prints 5
}
console.log(a) // prints 3
```

Const

```
// can be assigned only once
const a = 55
a = 44 // throws an error
```

Multiline string

```
console.log(`
 This is a
 multiline string
`)
```

Template strings

```
const name = 'World'
const message = `Hello ${name}`
console.log(message)
// prints "Hello World"
```

Exponent operator

```
const byte = 2 ** 8
// Same as: Math.pow(2, 8)
```

Spread operator

```
const a = [ 1, 2 ]
const b = [ 3, 4 ]
const c = [ ...a, ...b ]
console.log(c) // [1, 2, 3, 4]
```

String includes()

```
console.log('apple'.includes('pl'))
// prints true
console.log('apple'.includes('tt'))
// prints false
```

String startsWith()

```
console.log('apple'.startsWith('ap'))
//prints true
console.log('apple'.startsWith('bb'))
//prints false
```

String repeat()

```
console.log('ab'.repeat(3))
//prints "abababab"
```

Destructuring array

```
let [a, b] = [3, 7];
console.log(a); // 3
console.log(b); // 7
```

Destructuring object

```
let obj = {
 a: 55,
 b: 44
};
let { a, b } = obj;
console.log(a); // 55
console.log(b); // 44
```

Object property assignment

```
const a = 2
const b = 5
const obj = { a, b }
// Before es6:
// obj = { a: a, b: b }
console.log(obj)
// prints { a: 2, b: 5 }
```

Object.assign()

```
const obj1 = { a: 1 }
const obj2 = { b: 2 }
const obj3 = Object.assign({},
    obj1, obj2)
console.log(obj3)
// { a: 1, b: 2 }
```

Promises with finally

```
promise
 .then((result) => { ··· })
 .catch((error) => { ··· })
 .finally(() => { /* logic
independent of success/error */ })
/* The handler is called when the
promise is fulflled or rejected.*/
```

Object function assignment

```
const obj = {
 a: 5,
 b() {
 console.log('b')
 }
}
obj.b() // prints "b"
```

Object.entries()

```
const obj = {
 firstName: 'FirstName',
 lastName: 'LastName',
 age: 24,
 country: 'India',
};
const entries = Object.entries(obj);
/* returns an array of [key, value]
 pairs of the object passed */
console.log(entries);
/* prints
 [
['firstName', 'FirstName'],
['lastName', 'LastName'],
['age', 24],
['country', 'India']
]; */
```

Spread operator

```
const a = {
 firstName: "FirstName",
 lastName: "LastName1",
}
const b = {
 ...a,
 lastName: "LastName2",
 canSing: true,
}
console.log(a)
//{firstName: "FirstName", lastName: "LastName1"}
console.log(b)
/* {firstName: "FirstName", lastName: "LastName2",
canSing: true} */
/* great for modifying objects without side
effects/affecting the original */
```

Destructuring Nested Objects

```
const Person = {
 name: "Harry Potter",
 age: 29,
 sex: "male",
 materialStatus: "single",
 address: {
 country: "USA",
 state: "Nevada",
 city: "Carson City",
 pinCode: "500014",
 },
};
const { address : { state, pinCode }, name } = Person;
console.log(name, state, pinCode)
// Harry Potter Nevada 500014
console.log(city) // ReferenceError
```

JavaScript loops

@codingtute

for

```
for (let i = 0; i < 5; i++)
{
    console.log(i);
}
//Prints 0 1 2 3 4
```

do-while

```
let iterator = 0;
do {
  iterator++;
  console.log(iterator);
} while (iterator < 5);
//Prints 1 2 3 4 5
```

while

```
let iterator = 0;
while (iterator < 5) {
    iterator++;
    console.log(iterator);
}
//Prints 1 2 3 4 5
```

for...in

```
const arr = [3, 5, 7];
arr.foo = 'hello';
for (let i in arr) {
   console.log(i);
}
//Prints "0", "1", "2", "foo"
```

for...of

```
const arr = [3, 5, 7];
for (let i of arr) {
    console.log(i);
}
//Prints 3, 5, 7
```

JavaScript loops

@codingtute

for

```
for (let i = 0; i < 5; i++)
{
    console.log(i);
}
//Prints 0 1 2 3 4
```

do-while

```
let iterator = 0;
do {
  iterator++;
  console.log(iterator);
} while (iterator < 5);
//Prints 1 2 3 4 5
```

while

```
let iterator = 0;
while (iterator < 5) {
    iterator++;
    console.log(iterator);
}
//Prints 1 2 3 4 5
```

for...in

```
const arr = [3, 5, 7];
arr.foo = 'hello';
for (let i in arr) {
   console.log(i);
}
//Prints "0", "1", "2", "foo"
```

for...of

```
const arr = [3, 5, 7];
for (let i of arr) {
    console.log(i);
}
//Prints 3, 5, 7
```

JavaScript Arrays Destructuring

@codingtute

Assigning array items to variables

```
const [a, b, c] = [123, 'second', true];
// a => 123
// b => 'second'
// c => true
```

Skipping items

```
const [, b] = [123, 'second', true];
// b => 'second'
```

Assigning the first values, storing the rest together

```
const [a, b, ...rest] = [123, 'second', true, false, 42];
// a => 123
// b => 'second'
// rest => [true, false, 42]
```

Swapping variables

```
let x = true;
let y = false;
[x, y] = [y, x];
// x => false
// y => true
```

JavaScript Arrays Destructuring

@codingtute

Assigning array items to variables

```
const [a, b, c] = [123, 'second', true];
// a => 123
// b => 'second'
// c => true
```

Skipping items

```
const [, b] = [123, 'second', true];
// b => 'second'
```

Assigning the first values, storing the rest together

```
const [a, b, ...rest] = [123, 'second', true, false, 42];
// a => 123
// b => 'second'
// rest => [true, false, 42]
```

Swapping variables

```
let x = true;
let y = false;
[x, y] = [y, x];
// x => false
// y => true
```

JavaScript Async/Await

@codingtute

Async

When we append the keyword “async” to the function, this function returns the Promise by default on execution. Async keyword provides extra information to the user of the function:

- The function contains some Asynchronous Execution
- The returned value will be the Resolved Value for the Promise.

```
async function f() {
    return 1;
}
f().then(alert); // 1
```

Await

The keyword "await" makes JavaScript wait until that promise settles and returns its result

- The 'await' works only inside async functions

```
async function f() {
    let promise = new Promise((resolve, reject) => {
      setTimeout(() => resolve("done!"), 1000)
    });
    let result = await promise; // wait until the promise resolves
    alert(result); // "done!"
}
f();
```

JavaScript Async/Await

@codingtute

Async

When we append the keyword “async” to the function, this function returns the Promise by default on execution. Async keyword provides extra information to the user of the function:

- The function contains some Asynchronous Execution
- The returned value will be the Resolved Value for the Promise.

```
async function f() {
    return 1;
}
f().then(alert); // 1
```

Await

The keyword "await" makes JavaScript wait until that promise settles and returns its result

- The 'await' works only inside async functions

```
async function f() {
    let promise = new Promise((resolve, reject) => {
      setTimeout(() => resolve("done!"), 1000)
    });
    let result = await promise; // wait until the promise resolves
    alert(result); // "done!"
}
f();
```

JavaScript API Calls Cheat Sheet

@codingtute

XML HTTP Request

- All modern browsers support the XMLHttpRequest object to request data from a server.
- It works on the oldest browsers as well as on new ones.
- It was deprecated in ES6 but is still widely used.

```
var request = new XMLHttpRequest();
request.open('GET',
'https://jsonplaceholder.typicode.com/todos')
request.send();
request.onload = ()=>{

 console.log(JSON.parse(request.response));
}
```

Fetch

- The Fetch API provides an interface for fetching resources (including across the network) in an asynchronous manner.
- It returns a Promise
- It is an object which contains a single value either a Response or an Error that occurred.
- .then() tells the program what to do once Promise is completed.

```
fetch('https://jsonplaceholder.typicode.com/todos'
).then(response =>{
    return response.json();
}).then(data =>{
    console.log(data);
})
```

Axios

- It is an open-source library for making HTTP requests.
- It works on both Browsers and Node js
- It can be included in an HTML file by using an external CDN
- It also returns promises like fetch API

```
<script
src="https://cdn.jsdelivr.net/npm/axios/dis
t/axios.min.js"></script>

axios.get("https://jsonplaceholder.typicode
.com/todos")
.then(response =>{
    console.log(response.data)
})
```

jQuery AJAX

- It performs asynchronous HTTP requests.
- Uses $.ajax() method to make the requests.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery
/3.5.1/jquery.min.js"></script>

$(document).ready(function(){
    $.ajax({
        url:
"https://jsonplaceholder.typicode.com/todos",
        type: "GET",
        success: function(result){
            console.log(result);
        }
    })
})
```

JavaScript API Calls Cheat Sheet

@codingtute

XML HTTP Request

- All modern browsers support the XMLHttpRequest object to request data from a server.
- It works on the oldest browsers as well as on new ones.
- It was deprecated in ES6 but is still widely used.

```
var request = new XMLHttpRequest();
request.open('GET',
'https://jsonplaceholder.typicode.com/todos')
request.send();
request.onload = ()=>{

 console.log(JSON.parse(request.response));
}
```

Axios

- It is an open-source library for making HTTP requests.
- It works on both Browsers and Node js
- It can be included in an HTML file by using an external CDN
- It also returns promises like fetch API

```
<script
src="https://cdn.jsdelivr.net/npm/axios/dis
t/axios.min.js"></script>

axios.get("https://jsonplaceholder.typicode
.com/todos")
.then(response =>{
    console.log(response.data)
})
```

Fetch

- The Fetch API provides an interface for fetching resources (including across the network) in an asynchronous manner.
- It returns a Promise
- It is an object which contains a single value either a Response or an Error that occurred.
- .then() tells the program what to do once Promise is completed.

```
fetch('https://jsonplaceholder.typicode.com/todos'
).then(response =>{
    return response.json();
}).then(data =>{
    console.log(data);
})
```

jQuery AJAX

- It performs asynchronous HTTP requests.
- Uses $.ajax() method to make the requests.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery
/3.5.1/jquery.min.js"></script>

$(document).ready(function(){
    $.ajax({
        url:
"https://jsonplaceholder.typicode.com/todos",
        type: "GET",
        success: function(result){
            console.log(result);
        }
    })
})
```

JWT Token Generation

@codingtute

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. JSON Web Tokens consist of three parts separated by dots (.), which are: Header, Payload and, Signature. Therefore, a JWT typically looks like the following.

xxxxx.yyyyy.zzzzz

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Signature

```
HMACSHA256(
  base64UrlEncode(header) +
"." +
  base64UrlEncode(payload),
  secret)
```

base64UrlEncode

base64UrlEncode

```
eyJhbGciOiJIUz
I1NiIsInR5cCI6
IkpXVCJ9
```

+ "." +

```
eyJzdWIiOiIxMjM0
NTY3ODkwIiwibmFt
ZSI6IkpvaG4gRG9l
IiwiaWF0IjoxNTE2
MjM5MDIyfQ
```

+ "." +

```
SflKxwRJSMeKKF2QT
4fwpMeJf36POk6yJV
_adQssw5c
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

JWT Token Generation

@codingtute

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. JSON Web Tokens consist of three parts separated by dots (.), which are: Header, Payload and, Signature. Therefore, a JWT typically looks like the following.

xxxxx.yyyyy.zzzzz

Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Payload

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

Signature

```
HMACSHA256(
  base64UrlEncode(header) +
"." +
  base64UrlEncode(payload),
  secret)
```

base64UrlEncode

base64UrlEncode

```
eyJhbGciOiJIUz
I1NiIsInR5cCI6
IkpXVCJ9
```

+ "." +

```
eyJzdWIiOiIxMjM0
NTY3ODkwIiwibmFt
ZSI6IkpvaG4gRG9l
IiwiaWF0IjoxNTE2
MjM5MDIyfQ
```

+ "." +

```
SflKxwRJSMeKKF2QT
4fwpMeJf36POk6yJV
_adQssw5c
```

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

ES6 HTML DOM

@codingtute

Document properties of Legacy DOM

alinkColor: This property defines the color of the activated links.
Eg. document.alinkColor
anchors[]: It is the array of each anchor object, one for each anchor as it appears in the document.
Eg. document.anchors[0],document.anchors[1]
applets[]: It is the array of applet objects one for each applet as it appears in the document.
Eg. document.applets[0],document.applets[1]
bgColor: This property defines the background color of the document.
Eg. document.bgColor
Cookie: This property defines valued property with special behavior which allows the cookies associated with the document to be queried to set.
Eg. document.cookie
Domain: This property defines the domain that a document belongs to it has been used for security purpose.
Eg. document.domain
embeds[]: Synonym for plugins[]. It is the array of objects that represent data embedded in the document
Eg. document.embeds[0],document.embeds[1]
fgColor: This property defines the default text color for the document.
Eg. document.fgColor
forms[]: It is the array of forms object one for each, as it appears in the form.
Eg. document.forms[0],document.forms[1]
images[]: It is the array of form objects, one for each element with <img> tag as it appears in the form.
Eg. document.images[0],document.images[1]
lastModified: This property defines date of the most recent update.
Eg. document.lastModified
linkColor: This property defines the color of unvisited links it is the opposite of the vlinkColor.
Eg. document.linkColor
links[]: Document link array.
Eg. document.links[0],document.links[1]
Location: This property holds the URL of the document.
Eg. document.location
plugins[]: This property is the synonym for embeds[].
Eg.document.plugins[0],document.plugins[1]
Referrer: String that contains the URL of the document if it is linked with any.
Eg. document.referrer
Title: Contents of the <title> tag.
Eg. document.title
URL: This property defines the URL.
Eg. document.URL
vlinkColor: This property defines the color of the visited links(not-activated).
Eg. document.vlinkColor

Document methods in Legacy DOM

clear(): Erases the contents of the document and returns nothing.
Eg. document.clear()
close(): Closes the document opened with open().
Eg. document.close()
open(): Deletes the existing document content and opens a stream to which the new document contents may be written. Returns nothing.
Eg. document.open()
write(): Inserts the specified string in the document.
Eg. document.write()
writeln(): Same as write() but inserts a new line after it is done appending.
Eg. document.writeln()

Document properties in W3C DOM

body: Contents of the tag.
Eg. document.body
defaultView: Represents the window in which the document is displayed.
Eg. document.defaultView
documentElement: Reference to the tag of the document.
Eg. document.documentElement
implementation: Represents the DOMImplementation object that represents the implementation that created this document.
Eg. document.implementation

Documents methods in W3C DOM

createAttribute(name_of_attr): Returns a newly-created Attr node with the specified name.
Eg. document.createAttribute(name_of_attr)
createComment(text): Creates and returns a new Comment node containing the specified text.
Eg. document.createComment(some_text)
createDocumentFragment(): Creates and returns an empty DocumentFragment node.
Eg. document.createDocumentFragment()
createElement(tagname_of_new_ele): creates and returns a new Element node with a specified tagname.
Eg. document.createElement(tagname_of_new_ele)
createTextNode(text): Creates and returns a new Text node that contains the specified text.
Eg. document.createTextNode(text)
getElementById(Id): Returns the value from the document of the element with the mentioned Id.
Eg. document.getElementById(Id)
getElementsByName(name): Returns an array of nodes with the specified name from the document.
Eg. document.getElementsByName(name)
getElementsByTagName(tagname): Returns an array of all element nodes in the document that have a specified tagname.
Eg. document.getElementsByTagName(tagname)
importNode(importedNode, deep): Creates and returns a copy of a node from some other document that is suitable for insertion into this document. If the deep argument is true, it recursively copies the children of the node too.
Eg. document.importNode(importedNode, deep)

Document properties in IE4 DOM

activeElement: Refers to the currently active input element.
Eg. document.activeElement
all[]: An indexable array of all element objects within the document.
Eg. document.all[]
charset: Character set of the document.
Eg. document.charset
children[]: Array that contains the HTML elements that are the direct children of the document.
Eg. document.children[]
defaultCharset: Default charset of the document.
Eg. document.defaultCharset
expando: When this property is set to false, it prevents client side objects from getting expanded.
Eg. document.expando
parentWindow: Document containing window.
Eg. document.parentWindow
readyState: Specifies the loading status of the document.
Eg. document.readyState
uninitialized: Document has not yet started loading.
Eg. document.uninitialized
loading: Document is loading
Eg. document.loading
interactive: Document has loaded sufficiently for the user to interact.
Eg. document.interactive
complete: Document has loaded.
Eg. document.complete

Document methods in IE4 DOM

elementFromPoint(x,y): Returns the element located at the specified point.
Eg. document.elementFromPoint(x,y)