# **Javascript ES6+ Cheatsheet**

1. Conditional Statement
   1. iF - Else Statement

let status;

let data = [“element1”,“element2”,“element3”];

if (data.length == 0) {

status = "data null";

} else if (point == 1) {

status = "one value";

} else {

status = "many value";

}

* 1. Switch

switch (

new Date().getDay()) {

case 6: // if (day == 6)

text = "Saturday";

break;

case 0: // if (day == 0)

text = "Sunday";

break;

default:

// else...

text = "Whatever";

}

1. Variable

var varVariable = 10;

const constVariable = 10;

let letVariable = 10;

let a,b,c = 10;

1. Data Types

// Number: an integer or floating point

let numberInterType = 10;

let numberFloatType = 10.2;

// String: Contextual data

let stringType = "This is string";

// Boolean: Any of two value

let booleanTrueType = true;

let booleanFalseType = false;

// Object: Key-value pair collective data

let objectType = {

numberType: 10,

stringType: "This is string",

booleanType: true,

};

// Undefined: a variable which haven't been initialized

let undefinedType;

// Null: denotes a null value

let nullType = null

1. Arrays

let fruit = ["Banana", "Apple", "Pear"];

* 1. Array Methods

const java = ['j', 'a', 'v', 'a']

const script = ['s', 'c', 'r', 'i', 'p', 't']

// concat(): Join several arrays into one

const concat = concat()java.concat(script) //["j", "a", "v", "a", "s", "c", "r", "i", "p", "t"]

// indexOf(): Returns the first position at which a given element appears in an array

const indexOf = java.indexOf('a') // 1

// join(): Combine elements of an array into a single string and return the string

const join = java.join() // "j,a,v,a"

const join\_with\_seperator = java.join('-') // "j-a-v-a"

// lastIndexOf(): Gives the last position at which a given element appears in an array

const last\_index\_of = java.lastIndexOf('a') // 3

// pop(): Removes the last element of an array

java.pop() // java = ["j", "a", "v"]

// push(): Add a new element at the end

java.push('new value') // java = ["j", "a", "v", "a", "new value"]

// reverse(): Reverse the order of the elements in an array

java.reverse() // java = ["a", "v", "a", "j"]

// shift(): Remove the first element of an array

const removed\_value = java.shift() // j and java = ["a", "v", "a"]

// slice(): Pulls a copy of a portion of an array into a new array

const second\_to\_end = script.slice(1) // ["c", "r", "i", "p", "t"]

const second\_to\_fifth = script.slice(1, 5) // ["c", "r", "i", "p"]

// sort(): Sorts elements alphabetically

java.sort() // java = ["a", "a", "j", "v"]

// splice(): Adds elements in a specified way and position

java.splice(2, 0, 'One', 'two') // java = ["j", "a", "One", "two", "v", "a"]

const removeElements = java.splice(2, 1) // ["v"] and java = ["j", "a", "a"]

// unshift(): Adds a new element to the beginning

const new\_array\_length = java.unshift('One', 2) // 6 and java = ["One", 2, "j", "a", "v", "a"]

1. Operators
   1. Basic Operators

const addition = 1 + 1; // 2

const subtraction = 2 - 1; // 1

const multiplication = 2 \* 2; // 4

const division = 4 / 2; // 2

const grouping\_operator = 1 + (2 \* 3); // 7

const modulus\_or\_remainder = 5 % 2; // 1

const increment\_numbers = 5++; // 6

const decrement\_numbers = 5--; // 4

* 1. Comparison Operators

const equal\_to = (10 =='10')

const equal\_value\_and\_type = (10 === '10')

const not\_equal = (10 != 11)

const not\_equal\_value\_or\_type = (10 === '10')

const greater\_than = (10 > 11)

const less\_than = (10 < 11)

const greater\_than\_or\_equal\_to = (10 >= 11)

const less\_than\_or\_equal\_to = (10 <= 11)

const Ternary\_operator = (10 == 11 ?'true':'false')

* 1. Logical Operator

const logical\_and = (true && false)// false

const logical\_or = (true || false)// true

const logical\_not = (!false)// true

1. Global Function

// isNaN(): Determines whether a value is NaN (Not a Number)

isNaN(123) //false

isNaN('123') //false

isNaN('Hello') //true

// parseFloat(): Parses a string and returns a floating point number

let a = parseFloat("10") // 10

let b = parseFloat("10.00") // 10

let c = parseFloat("10.33") // 10.33

// parseInt(): Parses a string and returns an integer

let a = parseInt("10") // 10

let b = parseInt("10.00") // 10

let c = parseInt("10.33") // 10

// String(): Converts an object's value to a string

let x1 = String(false); // 'false'

let x2 = String(true); // 'true'

let x3 = String(new Date()); // 'Tue Jun 01 2021 22:29:05 GMT+0700 (Western Indonesia Time)'

let x4 = String("12345"); // '12345'

let x5 = String("12345"); // '12345'

// eval(): Evaluates a string and executes it as if it was script code

var a = eval("10 \* 10"); // 100

var b = eval("10 + 10"); // 20

1. Loop

let java = ['j', 'a', 'v', 'a']

let person = {firstname: 'Java', lastname: 'Script'}

// For: Loops through a block of code a number of times

let text = '' // Java

for (i = 0; i < java.length; i++) {

text += java[i]

}

// For In: Loops through the properties of an object

let text = ""; // Java Script

for (let x in person) {

text += person[x] + " ";

}

// For Of: Loops through the values of an iterable object

let text = '' // Java

for (let x of java) {

text += java[x]

}

// While: loops through a block of code while a specified condition is true

let evenNumbers = [] // [0, 2, 4, 6, 8]

let i = 0

while (i < 10) {

if (i%2 == 0) {

evenNumbers.push(i)

}

i++;

}

// Do While: Loops through a block of code while a specified condition is true

let evenNumbers = [] // [0, 2, 4, 6, 8]

let i = 0;

do {

if (i%2 == 0) {

evenNumbers.push(i)

}

i++;

}

while (i < 10);

// Break: Is used to jumps out of the loop

let text = '' // Ja

for (i = 0; i < java.length; i++) {

if (i === 2) { break; }

text += java[i]

}

// Continue: Is used to jumps over the loop

let text = '' // Jaa

for (i = 0; i < java.length; i++) {

if (i === 2) { continue; }

text += java[i]

}

1. String Methods

let java = "Java";

let script = "Script";

let javaScript = "Java Script"

// charAt: Returns the character at the specified index in a string

const firstCharacter = java.charAt(0); // J

// concat: Is used to join two or more strings.

const firstCharacter = java.concat(" ", script); // Java Script

// indexOf: Returns the position of the first occurrence of a specified value in a string

const index = java.indexOf(a); // 1

// lastIndexOf: Returns the position of the last occurrence of a specified value in a string.

const index = java.lastIndexOf(a); // 3

// replace: Returns a new string where the specified values are replaced.

const newJava = java.replace("Ja", 'newJa'); // newJava

// slice: Returns the extracted parts in a new string

const second\_to\_end = script.slice(1) // cript

const second\_to\_fifth = script.slice(1, 5) // crip

// split: Is used to split a string into an array of substrings, and returns the new array

const jsArray = javaScript.split(" "); // ["Java", "Script"]

// substr: Returns a portion of the string from given start and end index

const fifth\_to\_end = script.substr(5) // Script

const fifth\_to\_ninth = script.substr(5, 4) // Scri

// toLowerCase

const lowerCaseJava = java.toLowerCase() // java

// toUpperCase

const upperCaseJava = java.toUpperCase() // JAVA

1. Number Methods

// toFixed: Rounding to a specified number of decimals

const decimalNum = 1.23456;

const n = decimalNum.toFixed(2); // 1.23

1. Math Methods

const decimalNumber = 1.4

// ceil: Rounds a number UPWARDS to the nearest integer

const upWardsRounded = Math.ceil(decimalNumber) // 2

// floor: Rounds a number DOWNWARDS to the nearest integer

const downWardsRounded = Math.floor(decimalNumber) // 1

// max: returns the number with the highest value

const maxNumber = Math.max(5, 10) // 10

// min: returns the number with the lowest value

const minNumber = Math.min(5, 10) // 5

1. Dealing with Dates

const today = new Date()

const specificDate = new Date(2024, 6, 20, 10, 20, 30, 0)

const specificStringDate = new Date("2024-04-15")

* 1. Pulling Date and Time Values

const d = new Date(2024, 4, 15, 10, 20, 30, 0);

// Mon Apr 15 2024 10:20:30 GMT+0700 (Western Indonesia Time)

// getFullYear(): Get the year as a four digit number (yyyy)

const fullYear = d.getFullYear() // 2024

// getMonth(): Get the month as a number (0-11)

const month = d.getMonth() // 4

// getDate(): Get the day as a number (1-31)

const date = d.getDate() // 15

// getHours(): Get the hour (0-23)

const hour = d.getHours() // 10

// getMinutes(): Get the minute (0-59)

const minutes = d.getMinutes() // 20

// getSeconds(): Get the second (0-59)

const seconds = d.getSeconds() // 30

// getMilliseconds(): Get the millisecond (0-999)

const milliseconds = d.getMilliseconds() // 0

// getTime(): Get the time (milliseconds since January 1, 1970)

const time = d.getTime()

// getDay(): Get the weekday as a number (0-6)

const day = d.getDay()

1. Error

try {

// Block of code to try

throw "My error message"; // throw a text

}

catch(err) {

// Block of code to handle errors

}

finally { }

1. JSON(JavaScript Object Notation)

const jsonObject = {"firstName":"John","lastName":"Doe"};

const jsonArray = {

"employees":[

{"firstName":"John", "lastName":"Doe"},

{"firstName":"Anna", "lastName":"Smith"},

{"firstName":"Peter", "lastName":"Jones"}

]

};

// parse(): Convert JSON text into a JavaScript object

const jsonString = '{"firstName":"John", "lastName":"Doe"}';

const jsonObject = JSON.parse(jsonString) // {"firstName":"John", "lastName":"Doe"}

// stringify(): Convert JavaScript object into a JSON text

const jsonObject = {"firstName":"John", "lastName":"Doe"};

const jsonString = JSON.stringify(jsonString) // '{"firstName":"John", "lastName":"Doe"}'

## Arrow function

Arrow function trong JavaScript là hàm sử dụng kí hiệu => để định nghĩa, với cú pháp cơ bản như sau:

let func = (arg1, arg2,..., argN) => expression;

Trong đó: hàm trên nhận danh sách tham số là (arg1, arg2,..., argN), sau đó thực hiện biểu thức expression rồi return về kết quả của biểu thức expression

Nói cách khác, đây là cách biểu diễn rút gọn của biểu thức hàm:

let func = function(arg1, arg2,..., argN) {

return expression;

}

Cách sử dụng:

* Nhiều tham số

let func = (arg1, arg2,..., argN) => expression;

* Một tham số

let func = arg => expression;

* Không có tham số

let func = () => expression;

### Cú pháp với Object literal

let func = (id, name) =>({\_id: id, \_name: name});

* Arrow function với nhiều dòng code

let func = (id, name) => {

console.log(“user: id: ”,id, “ name: ”,name);

return {\_id: id, \_name: name};

}

1. Async/Await
   1. Khái niệm

**Async** được dùng để khai báo một hàm bất đồng bộ. Các hàm bất đồng bộ sẽ luôn trả về một giá trị. Việc sử dụng async chỉ đơn giản là ngụ ý rằng một lời hứa sẽ được trả lại và nếu một lời hứa không được trả lại, JavaScript sẽ tự động kết thúc nó.

**Await** được sử dụng để chờ một Promise. Nó chỉ có thể được sử dụng bên trong một khối Async. Từ khóa Await làm cho JavaScript đợi cho đến khi promise trả về kết quả. Cần lưu ý rằng nó chỉ làm cho khối chức năng không đồng bộ chờ đợi chứ không phải toàn bộ chương trình thực thi.

* 1. Cú pháp

### Async

async function myFunction() {

return "Hello";

}

hoặc

const myFunction = async () => {

return "Hello";

}

### Await

const myFunction = async () => {

// Từ khóa await được đặt trước một hàm được cho là hàm đợi một promise

let value = await promise;

return value;

}

* 1. Xử lý lỗi

Chúng ta có thể bắt lỗi đó bằng cách sử dụng try…catch, giống như cách thông thường throw:

async function f() {

try {

let response = await fetch('http://no-such-url');

} catch(err) {

alert(err); *// TypeError: failed to fetch*

}

}

### Async/Await thực hiện tuần tự

Ví dụ:

async function sequence() {

await promise1(50); *// Wait 50ms…*

await promise2(50); *// …then wait another 50ms.*

return "done!";

}

Đoạn code trên mất 100ms để hoàn thành, không phải một lượng thời gian lớn nhưng vẫn chậm. Điều này xảy ra là do nó đang diễn ra theo trình tự. Hai hàm được trả lại, cả hai đều mất 50ms để hoàn thành. Hàm thứ hai chỉ thực hiện sau khi hàm đầu tiên được giải quyết.

Đối với một số trường hợp các yêu cầu lớn có thể rất tốn thời gian. Chúng ta phải thực hiện song song.

Chúng ta có thể làm được điều đó bằng cách sử dụng Promise.all()

async function sequence() {

await Promise.all([promise1(), promise2()]);

return "done!";

}

Hàm Promise.all() giải quyết khi tất cả các lời hứa bên trong có thể lặp được giải quyết và sau đó trả về kết quả.

Theo MDN: “Phương thức Promise.all(iterable) trả ra một Promise mới và promise mới này chỉ được kết thúc khi tất cả các promise trong iterable kết thúc hoặc có một promise nào đó xử lý thất bại. Kết quả của promise mới này là một mảng chứa kết quả của tất cả các promise theo đúng thứ tự hoặc kết quả lỗi của promise gây lỗi.”