JavaScript Cheatsheet

Item	Syntax	Description	Example

Declaring Variables var, let, const	`let < var_name > = < value >`	**var** - global access, value can chage
		
		let - access within block where it is declared, value can change
		
		const - access within block where it is declared, value cannot change
			`let i = 5;`
			
			`var myStr = "John";`
			
			`const pi = 3.14`

Strings			

length	*string_obj*.length	**length** Returns the length of the string	`let myStr = "Hello";`
			`console.log(myStr.length);`
			
			Output is 5

split	*string_obj*.split(*separator*)	**split** Splits the string based on the separator and returns an array.	`let myStr = "Hello! How are you?";`
			`console.log(myStr.split(" "))`
			
			Output is ['Hello!', 'How', 'are', 'you?']

charAt	*string_obj*.charAt(*index*)	**charAt** returns the character at a specified index in a string. Index starts at 0 ends at length-1	`let myStr = "Hello";<`
			`console.log(myStr.charAt(0))`
			
			Output is H

replace	*string_obj*.replace(*"SearchValue","NewValue"*)	**replace** searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced.	`let myStr = "Hello User";`
			`console.log(myStr.replace("User","World"));`
			
			Output is Hello World

substring	*string_obj*.substring(start, end)	**substring** is used to extract characters, between to indices from the given string, and returns the	`let myStr="Hello";`
			`console.log(myStr.substring(1,4));`
			
			Output is ell

		substring. It excludes the last index	

startswith	*string_obj*.startsWith(searchvalue)	**startsWith** returns true if a string begins with a specified string, otherwise false	`let myStr="Hello from the other side";`
			`console.log(myStr.startsWith("Hello"));`
			
			Output is *true*

endsWith	*string_obj*.endsWith(searchvalue))	**endsWith** returns true if a string ends with a specified string, otherwise false	`let myStr="Hello from the other side";`
			`console.log(myStr.startsWith("side"));`
			
			Output is *true*

toUpperCase	*string_obj*.toUpperCase()	**toUpperCase** converts a string to uppercase letters	`let myStr="hello";`
			`console.log(myStr.toUpperCase());`
			
			Output is HELLO

toLowerCase	*string_obj*.toLowerCase()	**toLowerCase** converts a string to lowercase letters	`let myStr="HELLO";`
			`console.log(myStr.toUpperCase());`
			
			Output is hello

concat	*string_obj*.concat(*string1, string2,..,stringN*)	**concat** joins two or more strings.	`let myStr="Hello"; let str="World";`
			`console.log(myStr.concat(str));`
			
			Output is HelloWorld

		Arrays	

push	*arr_name*.push(*value*)	**push** adds new items to the end of an array.	`let myArr=["Hello"]; myArr.push("World");`
			`console.log(myArr);`
			
			Output is [“Hello”,”World”]

pop	*arr_name*.pop()	**pop** removes the last element of an array.	`let myArr=["Hello","World"]; myArr.pop();`
			`console.log(myArr);`
			
			Output is [“Hello”]

length	*arr_name*.length	**length** sets or returns the number of elements in an array.	`let myArr=["Hello","World"];`
			`console.log(myArr.length);`
			
			Output is 2

indexOf	*arr_name*.indexOf(*item*)	**indexOf** searches for a specified item and returns its position.	`let myArr=["Hello","World"];`
			`console.log(myArr.indexOf("World")`
			
			Output is 1

lastIndexOf	*arr_name*.lastIndexOf(*item*)	**lastIndexOf** returns the last index (position) of a specified value.	`let myArr=["Hello","World","Hello"];`
			`console.log(myArr.lastIndexOf("Hello");`
			
			Output is 2

entries	*arr_name*.entries()	**entries** Returns and Array Iterator that helps you to iterate through the array and recieve each entry as an	`const hello = ["h", "e", "l", "l","o"];`
			`console.log(hello.entries());`
			
			Output is
			Object [Array Iterator] {}

		array of two elements containing the key and the value, where in the key is the index position of the element and value is the element itself.
find	`Array.find(<arrElemet>=>{ //return boolean based on a condition }`	**find** Finds the first occurance of an element in the array which returns true on checking the condition	`//Find the first string with s let myarr = ["Mercury","Venus","Earth","Mars"]; let found = myarr.find(val=>{ return val.includes("s"); }) console.log(found);` Output Venus
filter	`Array.filter(<arrElemet>=>{ //return boolean based on a condition }`	**filter** Finds the all occurances of elements in the array which returns true on checking the condition	`//Find the all strings with s let myarr = ["Mercury","Venus","Earth","Mars"]; let found = myarr.filter(val=>{ return val.includes("s"); }) console.log(found);` Output [Venus,Mars]

map	`Array.map(<arrElemet>=>{ //return processed value }`	**map** Processes the all elements of the array which returns a new processed array of same size	`let myarr = ["name","place","thing","animal"]; let found = myarr.map(val=>{ return val+"s"; }) console.log(found);`
			
			`Output [ 'names', 'places', 'things', 'animals' ]`

concat	*arr_name*..`concat(arr1.name);`	**concat** concatenates (joins) two or more arrays.	`let hello = ["hello", "world" ]; let lorem = ["along","lorem"] let h = hello.concat(lorem); console.log(h);` Output is ["hello", "world", "along", "lorem"]
		Map	
set	`mapName.set(key,value);`	**set** helps you define a new element with akey and its value	`var newMap = new Map(); newMap.set("h", 1); console.log(newMap);` Output is {"h" => 1}
get	`mapName.get(key);`	**get** helps you return a value of key you are searching for	`var newMap = new Map(); newMap.get("h"); console.log(newMap);` Output is Map(0) {size: 0}
keys	`mapName.keys();`	**get** is used to get all of the keys associated with the mapName	`var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.keys());` Output is {"h", "i"}
values	`mapName.values();`	**values** is used to get all of the values to the keys associated with the mapName	`var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.values());` Output is {1,2}

has	`mapName.has(key_name);`	**has** is used to check if the key passed resides in the map or not, and returns true or false	`var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); console.log(newMap.has(i));` Output is true
delete	`mapName.delete(key_name);`	**delete** is used to delete the key and the value from the map	`var newMap = new Map(); newMap.set("h",1); newMap.set("i",2); newMap.delete("h"); console.log(newMap);` Output is {"i" => 2}
		JSON	
Create JSON	`let varname={name1:value1,name2:values2,.....}`	JSON is a dictionary Object with Key-Value pairs.	`let myjson1={}; let myjson2 = {"name":"Jennifer","age":"32"}`
Add entry to JSON	`let jsonObj[<key>]=<value>`	Adds an entry to JSON Object mapping the key to value	`let myjson1 = {}; myjson1["name"]="Jason"; console.log(myjson1);`
		Operators	
Arithmetic	`<Operand1> <Operator> <Operand2>`	+ addition - subtration / division * multiplication % modulus(gives remainder) ++ increment by 1 – decrement by 1	`let num1 = 2; let num2 = 2; console.log(num1+num2); console.log(num1-num2); console.log(num1/num2); console.log(num1*num2); console.log(num1%num2); num1++; console.log(num1); num2--; console.log(num1);` Output is 4 0 1 4 0 3 3
Logical	`condition1 && condition2 condition1 || condition2 ! condition1`	**&&** (AND)is used to check if all the operand conditions are true || (OR)is used to check if either of the operand condition are true **!** (NOT) is used to check if the operand condition is not met	`let num1 = 12, num2 = 2; console.log(num1>10 && num2>10); console.log(num1>10 || num2>10); console.log(!(num1==num2));` Output is false true true
Assignment	`variable = value variable += incremental value variable -= decremental value %= modulus value /= divide value *= multiply value`	**a=b** assigns the value of b to a **a+=b** adds the value of b to a and stores it in a	`let num1 = 12, num2 = 2; console.log(num1=num2); console.log(num1+=num2); console.log(num1-=num2); console.log(num1/=num2); console.log(num1*=num2); console.log(num1%num2); console.log(num1=num2);` Output is 2 14 10 6 24 0 2

		a-=b subtracts the value of b from a and stores it in a **a%=b** divides the value of a by b and stores the remainder in a **a/=b** divides the value of a to b and stores the quotient in a **a*=b** multiplies the value of a and b and stores the value in a	
Loops			
For Loop	`for(initialization;condition;increment/decrement) { //code block }`	**for** loops throughout the block of code a number of times making sure the condition is satisfied	`for(let num = 0 ; num <=5 ; num++){ console.log(num) }` Output is 0 1 2 3 4 5
while	`while(condition){ //code block }`	**while** itrates through the block of code while a specified condition is true	`let num1 = 0; let num2 = 5; while(num1 < num2){ console.log(num1) num1++; }` Output is 0 1 2 3 4

do while	`do{ //code block } while(condition)`	**do while** loops throughout the block once before checking condition.	`let num = 5; do { console.log(num); num--; }`
			`while(num > 0)`
			
			Output is 5 4 3 2 1

for in	`for (var in object) { //code block }`	**for in** is used to itrate through the specific property/type of the object	`let arr = ["a","b","c"]; for(let i in arr) { console.log(arr[i]); }` Output is a b c
Conditional statements			
if	`if(condition){ //code Block... }`	**if** a specified condition is true, a block of code will be executed	`let num = 5; if(num = 5){ console.log(true); }` Output is true
if-else	`if(condition){ //Code Block... } else { //Code Block... }`	**if** a specified condition is true, a block of code will be executed. in case of false, else block is executed	`let num = 5; if(num = 4){ console.log(true) } else { console.log(false) }` Output is false

if-else if-else	`if(condition){ //Code Block... } else if (condition) { //Code Block... } else { //Code Block... }`	**else if** to specify a new condition to test, if the first/previous condition is false	`let num = 10; if(num < 10){ console.log("number is smaller"); } else if(num = 10) { console.log("number is equal"); } else { console.log("number is greater"); }` Output is number is equal
switch	`switch(expression) { case <value1>: //code break; case <value2>: //code break; . . . default: //default code block }`	**switch** to select one of many blocks of code to be executed. And **break** is used to end the preocessing within the switch statement.	`let num = 2; switch(num) { case 1: console.log("Hello world!"); break; case 2: console.log("Hi"); break; default: console.log("this is default"); }` Output is Hi
Other useful operations			
typeof	`typeof(operand)`	**typeof** operator returns a string indicating the type of the unevaluated operand	`console.log(typeOf("Hello"))` Output is "string"
isNaN	`isNaN(operand)`	**isNaN** determines whether a value is anythying but a number or not. It returns false for a number	`console.log(isNaN("Hello"))` Output is true
parseInt	`parseInt(string, radix)`	**parseInt** is a function that parses a string argument and returns an integer of the specified radix.(radix is a base)	//0011 is 3 for binary, since binary only has 2 numbers 0, 1 the radix is 2 `console.log(parseInt("0011", 2)); //Default parseInt takes decimal system console.log(parseInt("54"));` Output is 3 54
parseFloat	`parseFloat(string)`	**parseFloat** is a function that parses a string argument and returns an float	`parseFloat("3.14")` Output is 3.14

This cheatsheet covers the JS you will mostly use. To learn more commands you can go to this link.

Changelog

Date	Version	Changed by	Change Description
25-09-2021	1.0	Lavanya T S	Initial version created