JavaScript Cheat Sheet. # 100Days Of CodewithLux

Javascript is one of most popular programming language for the web. It is widely used for creating interactive client-side web applications and games. Javascript can be also used as server-side programming language. It is dynamic, weakly typed, prototype-based and multi-paradigm high-level programming language. With Introduction of Frameworks and libraries Like React, Angular, VueJS, node among others also MEAN and MERN major stacks for Full Stack JavaScript developers. Demand for JavaScript developers has exponentially increased in the past years; Theses developers may be Front end engineers, Back end engineers and systems Engineers.

1). Seven (7) Types.

1. String - "Any text"
2. Number - 33.54
3. Boolean - true or false
4. Null - null
5. Undefined - undefined
6. Symbol - Symbol('something')
7. Object - { key: 'value'}
   - Array - [1, "text", false]
- Function - function name() { }

2). Basic Vocabulary.

a). Variable - A named reference to a value is a variable.

```
var a = 7 + "2";
```

b). Operator - Operators are reserved-words that perform action on values and variables.

Examples: + - = * in === typeof != …

c). Keyword / reserved word - Any word that is part of the vocabulary of the programming language is called a keyword (a.k.a reserved word).

Examples: var = + if for…

d). Statement - A group of words, numbers and operators that do a task is a statement.

e). Expression - A reference, value or a group of reference(s) and value(s) combined with operator(s), which result in a single value.

3 Object

An object is a data type in JavaScript that is used to store a combination of data in a simple key-value pair.

```
var user = {
name: "Ghat doe ",
yearOfBirth: 1997,
calculateAge: function(){
    return 2000 – yearOfBirth;
}
}
```

a). Key -These are the keys in user object. In the above example name is a key.

b). Value - These are the values of the respective keys in user object 1997 is a value.

c). Method -If a key has a function as a value, its called a method.

4. Function.

A function is simply a bunch of code bundled in a section. This bunch of code ONLY runs when the function is called. Functions allow for organizing code into sections and code re-usability.

Using a function has ONLY two parts:
1) . Declaring/defining a function.
2). Calling/running a function.

```
// Function declaration / Function statement
function someName(param1, param2){
// bunch of code as needed…
var a = param1 + "love" + param2;
return a;
}
// Invoke (run / call) a function
someName("Me", "You")
```

a). Name of function
It's just a name you give to your function.
Tip: Make your function names descriptive to what the function does.

b). Parameters / Arguments (optional)
A function can optionally take parameters (a.k.a arguments). The function can then use this information within the code it has.

c). Code block.

Any code within the curly braces { ... } is called a "block of code", "code block" or simply "block". This concept is not just limited to functions. "if statements", "for loops" and other statements use code blocks as well.

d). Return (optional)
A function can optionally spit-out or "return" a value once its invoked. Once a function returns, no further lines of code within the function run.

e). Invoke a function
Invoking, calling or running a function all mean the same thing. When we write the function name, in this case someName, followed by the brackets symbol () like this someName(), the code inside the function gets executed.

f). Passing parameter(s) to a function (optional)
At the time of invoking a function, parameter(s) may be passed to the function code.

5 Vocabulary around variables and scope.

a). Variable Declaration - The creation of the variable.

```
let Age;
```

b). Variable Initialization - The initial assignment of value to a variable.

```
Age =22;
```

b). Variable Assignment- Assigning value to a variable.

Name = “ghat”;

c). Hoisting Variables- are declared at the top of the function automatically, and initialized at the time they are run.

```
console.log(a);
let a = "me";
```

d). Scope - The limits in which a variable exists.

e). Global scope - The outer most scope is called the Global scope.

f). Functional scope - Any variables inside a function is in scope of the function.

g) Lexical Environment (Lexical scope)- The physical location (scope) where a variable or function is declared is its lexical environment (lexical scope).

Rule:

1). Variables in the outer scope can be accessed in a nested scope; But variables inside a nested scope CANNOT be accessed by the outer scope. (a.k.a private variables.)

2). Variables are picked up from the lexical environment.

Scope chain - The nested hierarchy of scope is called the scope chain. The JS engine looks for variables in the scope chain upwards (it its ancestors, until found).

```
var a = "global";
function first(){
 let a = "fresh";
function second(){
 console.log(a); }
}
```

6 Coercion

When trying to compare different "types", the JavaScript engine attempts to convert one type into another so it can compare the two values.

Type coercion priority order:

1). String.

2). Number.

3. Boolean

```
3 + "2";
```

which returns - "32"

7 Conditional Statements.

1). If -else Statement: -

Run certain code, "if" a condition is met. If the condition is not met, the code in the "else" block is run (if available.)

2). Switch Statement: -

Takes a single expression, and runs the code of the "case" where the expression matches. The "break" keyword is used to end the switch statement.

8 Truthy / Falsy-

There are certain values in JavaScript that return true when coerced into boolean. Such values are called truthy values. On the other hand, there are certain values that return false when coerced to boolean. These values are knows as falsy values.

Ternary Operator: A ternary operator returns the first value if the expression is truthy, or else returns the second value.

9 Loop Statements-

Loops are used to ,m do something repeatedly. For instance lets say we get a list of 50 blog posts from the database and we want to print their titles on our page. Instead of writing the code 50 times, we would instead use a loop to make this happen.

Ways to create a variable

There are 3 ways to create variables in JavaScript: var, let and const. Variables created with var are in scope of the function (or global if declared in the global scope); let variables are block scoped; and const variables are like let plus their values cannot be re-assigned.

```
var a = "some value"; // functional or global scoped
let b = "some value"; // block scoped
const c = "some value"; // block scoped + cannot get new value
```

10 Promise

What is a Promise? Promise is an object that provides a useful construct when dealing with asynchronous tasks. A promise is called a "Promise" because it guarantees it will run upon success or failure of that task.

Working with a promise consists of two parts;

A). Creating a promise.

B). Using a promise.

```
// (A) Create a promise
const p = new Promise((resolve, reject)=>{
// Do some async task
 setTimeout(()=>{
 if(condition){
 resolve('Successful login');
} else {
reject('Login failed'); }
 }, 2000)
})
```

What is an Async task?

An async task is one in which a third-party process is doing the task.

Examples:

- Requesting/sending data to a database

- Requesting/sending data via HTTP protocol

- Working with the file system of the computer

```
// (B) Using a promise
p.then((res)=>{
 console.log(res)
}) .catch((err)=>{
```

```
console.log(err)
})
```

Note:

90% of the time you will be working with pre-existing promises. The step of "Creating a promise" would be done for you either by a library, framework or environment you are using. Examples of promises: fetch

11 Constructor.

What is a constructor?

In JavaScript, a constructor is a special function that acts as a mold to create new objects.

There are numerous built-in constructors in JavaScript, such as String, Number, Promise, Date, Array, Object, and many more.

We can create our own custom constructors if need be.

A great place to use a constructor is when you are creating multiple objects of the same kind.

There are two parts to working with a constructor:

1). Defining a constructor When creating a custom constructor

2). Using a constructor with the "new" keyword

Rule of thumb:

A) Set properties inside a constructor.

B) Set methods inside the prototype property.

```
//Defining a Constructor
function Car(make, model, year){
this.make = make;
this.model = model;
this.year = year;
this.setMiles = function(miles){
this.miles = miles return miles;
```

```
} }
// Using a constructor
const car1 = new Car('Toyota', 'Prius', 2016);
const car2 = new Car('Hyundai', 'Sonata', 2018);
// Adding method to the constructor
 prototype Car.prototype.age = function(){
 return (new Date()).getFullYear() - this.year;
}
car1.age(); // 2
```

In the above snippets:

"new" keyword The new keyword is used to create a new object (instance) from the constructor.

"prototype" property prototype is a special property on every object. Properties (methods or values) attached to the prototype property get inherited to every instance of the constructor.

12. 'this' keyword

The this keyword is used inside a function. The this keyword is merely a reference to another object.

What the this keyword refers to depends on the scenario or the way the function is implemented.

Here are the 3 scenarios to remember:

Scenario #1: this inside a function

The this keyword points to global object.

Scenario #2: this inside a method

The this keyword points to the object the method is in.

Scenario #3: When function is run with call, bind or apply

When a function is called using the .call(param) .bind(param) or .apply(param) method, the first param become the object that the this keyword refers to.

```
var name = "Ghat";
 function fun(){
 // some code here
console.log(this.name);
}
const user = {
name: "John Doe",
 yearOfBirth: 1999,
 calcAge: function(){
     const currentYear = (new Date()).getFullYear();
   return currentYear - this.yearOfBirth;
 }
 }
 fun();   // 'this' is global. Logs "Ghat"
user.calcAge();   // 'this' is the user object
fun.call(user);  // 'this' is the user object. Logs "John Doe"
```

Please feel free to share this PDF with anyone for free

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk Please send feedback and corrections to web

mbaabuharun8@gmail.com