JavaScript Cheatsheet

Learn JavaScript Correctly (Video course) https://ilovecoding.org/courses/js2

1 Seven (7) Types

Six Primitive Types

1. String — `"Any text"`
2. Number — `123.45`
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

Variable
A named reference to a value is a variable.

Operator
Operators are reserved-words that perform action on values and variables.
Examples: `+ - = * in === typeof !=` ...

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

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

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...`

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. Thats it.

```
var user = {
  name: "Aziz Ali",
  yearOfBirth: 1988,
  calculateAge: function(){
    // some code to calculate age
  }
}
```

Key
These are the keys in user object.

Value
These are the values of the respective keys in user object.

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 reusability.

Using a function has ONLY two parts. (1) Declaring/defining a function, and (2) using/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")
```

Name of function
Thats it, its just a name you give to your function. Tip: Make your function names descriptive to what the function does.

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.

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.

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.

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.

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

```
var a;
```

Variable Declaration
The creation of the variable.

```
a = 12;
```

Variable Initialization
The initial assignment of value to a variable.

```
a = "me";
```

Variable Assignment
Assigning value to a variable.

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

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

Scope
The limits in which a variable exists.

Global scope
The outer most scope is called the Global scope.

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

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.

```
var a = "global";

function first(){
    var a = "fresh";

    function second(){
        console.log(a);
    }
}
```

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)

JavaScript Cheatsheet

Learn JavaScript Correctly (Video course) https://ilovecoding.org/courses/js2

6 Operators

Full list of JavaScript operators https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators

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

Arithmetic
.. + .. Add
.. - .. Subtract
.. * .. Multiply
.. / .. Divide
.. % .. Remainder
.. ** .. Exponential

Assignment
.. = .. Assign value
.. += .. Add then assign
.. -= .. Subtract then assign
.. *= .. Multiply then assign

Logical
.. || .. Or
.. && .. And

Equality
.. === .. Equality
.. == .. Equality with coercion

Conversion
+ .. Convert to number
- .. Convert to number then negate it
! .. Convert to boolean then inverse it

Relational / Comparison
.. >= .. Greater than or equal to
.. <= .. Less than or equal to
.. != .. Not equal after coercion
.. !== .. Not equal

Increment / Decrement
..++ Postfix increment
..-- Postfix decrement

++.. Prefix increment
--.. Prefix increment

Others
typeof ..
.. instanceof ..
(..)
...spread-operator
.
..[..]
new ..
delete ..
(.. ? .. : ..)

Operator Precedence
Given multiple operators are used in an expression, the "Operator Precedence" determines which operator will be executed first. The higher the precedence, the earlier it will get executed.

Operator Associativity
Given multiple operators have the same precedence, "Associativity" determines in which direction the code will be parsed.

See the **Operator Precedence and Associativity table** here:
http://bit.ly/operatortable

7 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

```
2 + "7";   // "27"
true - 5   // -4
```

Coercion in action
Does this make sense?

8 Conditional Statements

Conditional statements allow our program to run specific code only if certain conditions are met. For instance, lets say we have a shopping app. We can tell our program to hide the "checkout" button if the shopping cart is empty.

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.)

```
if (a > 0) {
  // run this code
} else if (a < 0) {
  // run this code
} else {
  // run this code
}
```

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

```
(expression)? ifTrue: ifFalse;
```

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.

```
switch (expression) {
  case choice1:
    // run this code
    break;

  case choice1:
    // run this code
    break;

  default:
    // run this code
}
```

9 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.

Truthy Values

```
true
"text"
72
-72
Infinity
-Infinity
{}
[]
```

Falsy Values

```
false
""
0
-0
NaN
null
undefined
```

10 Loop Statements

Loops are used to 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.

For loop

```
for (initial-expression; condition; second-expression){
  // run this code in block
}
```

Step 1: Run the initial expression.

Step 2: Check if condition meets. If condition meets, proceed; or else end the loop.

Step 3: Run the code in block.

Step 4: Run the second-expression.

Step 5: Go to Step 2.

While loop

```
while (i<3){
  // run this code in block
  i++;
}
```

Step 1: If the condition is true, proceed; or else end the loop.

Step 2: Run the code in block.

Step 3: Go to Step 1.

Do while loop

```
do {
  // run this code in block
  i++;
} while (i<3);
```

Step 1: Run the code in block.

Step 2: If the condition is true, proceed; or else end the loop.

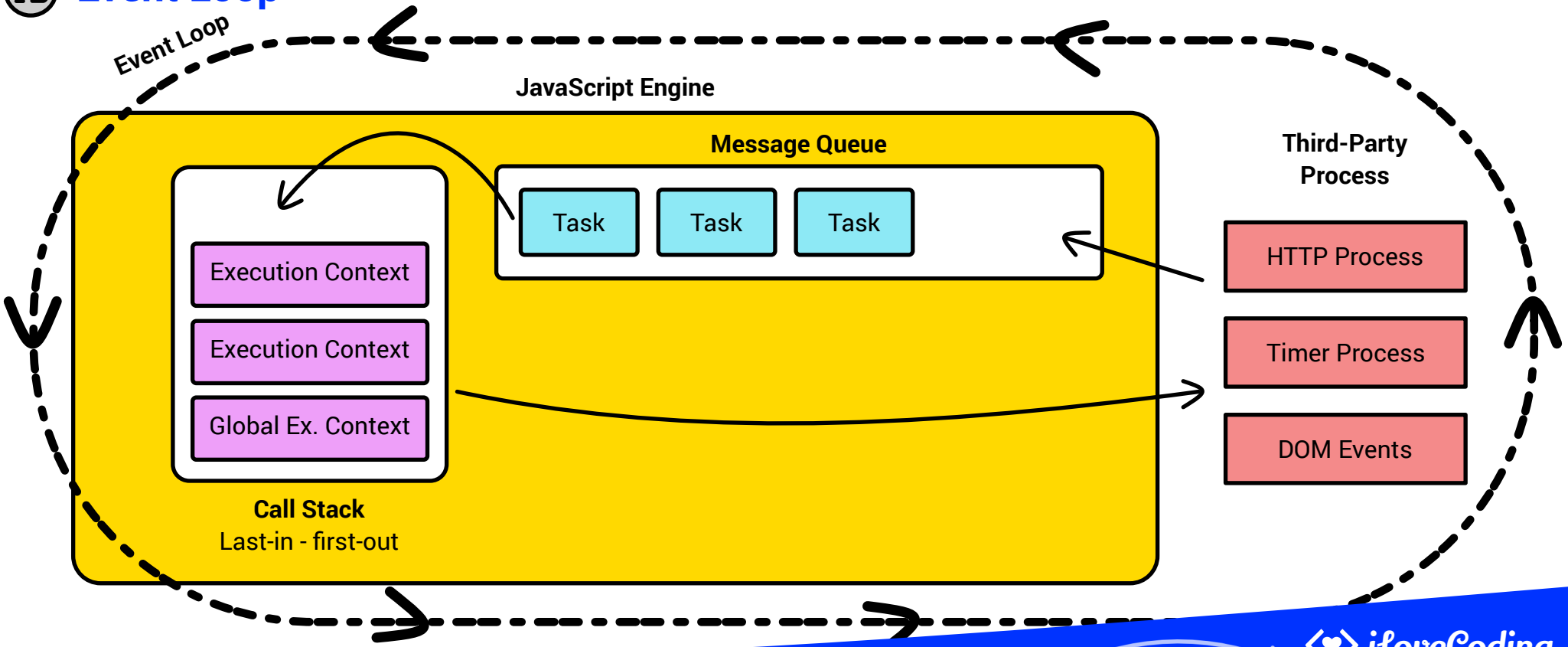
Step 3: Go to Step 1.

11 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
```

12 Event Loop



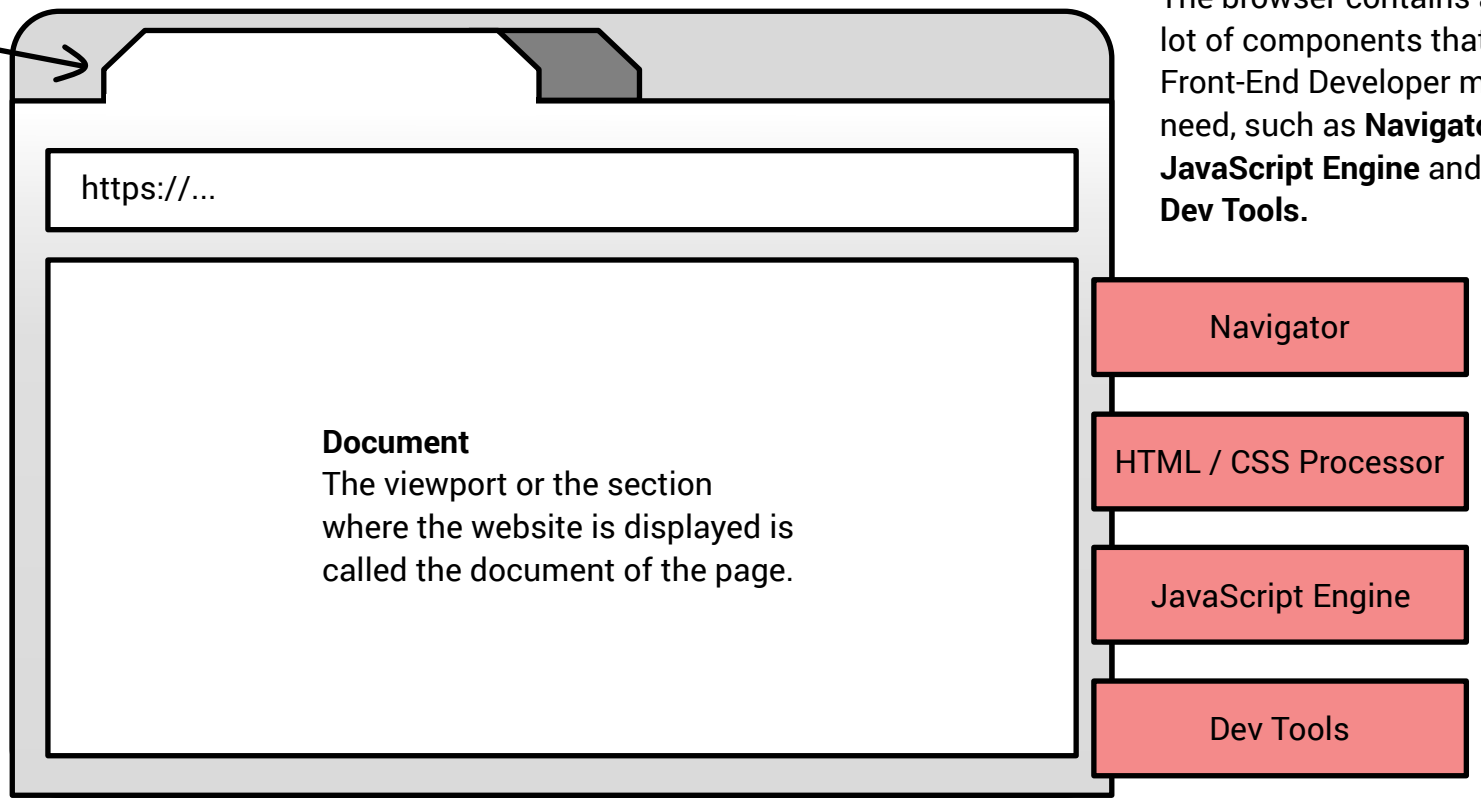
13 Browser

A web browser is a pretty advance piece of software which contains a lot of components. Many of these components are accessible to a web developer, so we can create complex web apps. At the same time a lot of components are kept out of reach of the web developer for security purposes. For instance, we as web developers can get access to the user's location, but we cannot get access to the user's saved passwords or browsing history. **Let's see below how a browser is structured:**

Window
Each tab of a browser is considered the window.

This is the outer most container that a web-app can access.

Notice: A website opened in one tab CANNOT access the window object of another tab. Pretty cool right?



The browser contains a lot of components that a Front-End Developer may need, such as **Navigator**, **JavaScript Engine** and **Dev Tools.**

14 DOM - Document Object Model

Query/Get Elements

```
// Preferred way:
document.querySelector('css-selectors')
document.querySelectorAll('css-selectors', ...)

// Old ways, and still work:
document.getElementsByTagName('element-name')
document.getElementsByClassName('class-name')
document.getElementById('id')
```

Create / clone Element

```
document.createElement('div')
document.createTextNode('some text here')
node.cloneNode()
node.textContent = 'some text here'
```

Add node to document

```
parentNode.appendChild(nodeToAdd)
parentNode.insertBefore(nodeToAdd, childNode)
```

Get Element Details

```
node.nextSibling
node.firstChild
node.lastChild
node.parentNode
node.childNodes
node.children
```

Modify Element

```
node.style.color = 'red'
node.style.padding = '10px',
node.style.fontSize = '200%'

node.setAttribute('attr-name', 'attr-value')
node.removeAttribute('attr-name')
```

Get and Modify Element Class

```
node.classList
node.classList.add('class-name', ...)
node.classList.remove('class-name', ...)
node.classList.toggle('class-name')
node.classList.contains('class-name')
node.classList.replace('old', 'new')
```

Remove Node

```
parentNode.removeChild(nodeToRemove)
// Hack to remove self
nodeToRemove.parentNode.removeChild(nodeToRemove)
```

Events

```
node.addEventListener('event-name', callback-function)
node.removeEventListener('event-name', callback-function)
```

List of Events: https://developer.mozilla.org/en-US/docs/Web/Events
or google *"Mozilla event reference"*

What is a "Node"? (in the context of DOM)

Node: Every item in the DOM tree is called a node. There are two types of node - A text node, and an element node:

Text Node: Node that has text.

Element Node: Node that has an element.

Child Node: A node which is a child of another node.

Parent Node: A node which has one or more child.

Descendent Node: A node which is nested deep in the tree.

Sibling Node: A node that share the same parent node.