Learnathon 2.0

Class -2

Topics List

1) JS variables and types
2) JS Control Flow (Looping)
3) Functions, Arrow Functions, Objects and Array
4) JS scooping, hoisting
5) Execution Context
6) Polyfilling
7) This keyword in JS

Safwan Alamgir
Sr. Software Engineer L-II
Vivasoft Limited

JS variables and types

We can declare variables in JavaScript in three ways:

1) Using var
2) Using let
3) Using const

Feature	var	let	const
Stored in Global Scope	Yes	No	No
Function Scoped	Yes	Yes	Yes
Block Scoped	No	Yes	Yes
Reassignable	Yes	Yes	No
Redeclarable	yes	No	No
Can be Hoisted	Yes	No	No
Hoisting Behavior	Initialized with undefined	Uninitialized	Uninitialized

JS variables and types

JavaScript Variable Nameing Conventions

- These are case-sensitive
- You should not use any of the JavaScript reserved keywords as a variable name.
- Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign

Reserved Words

Here is a list of reserved words (*and words to avoid*) in JavaScript:

abstract
boolean
break
byte
case
catch
char
class
continue
const
debugger
default
delete
do
double
else
enum
export
extends
false
final
finally
float
for
function
goto
if
implements
import
in
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
super
switch
synchronized
this
throw
throws
transient
true
try
typeof
var
void
while
with

JS variables and types

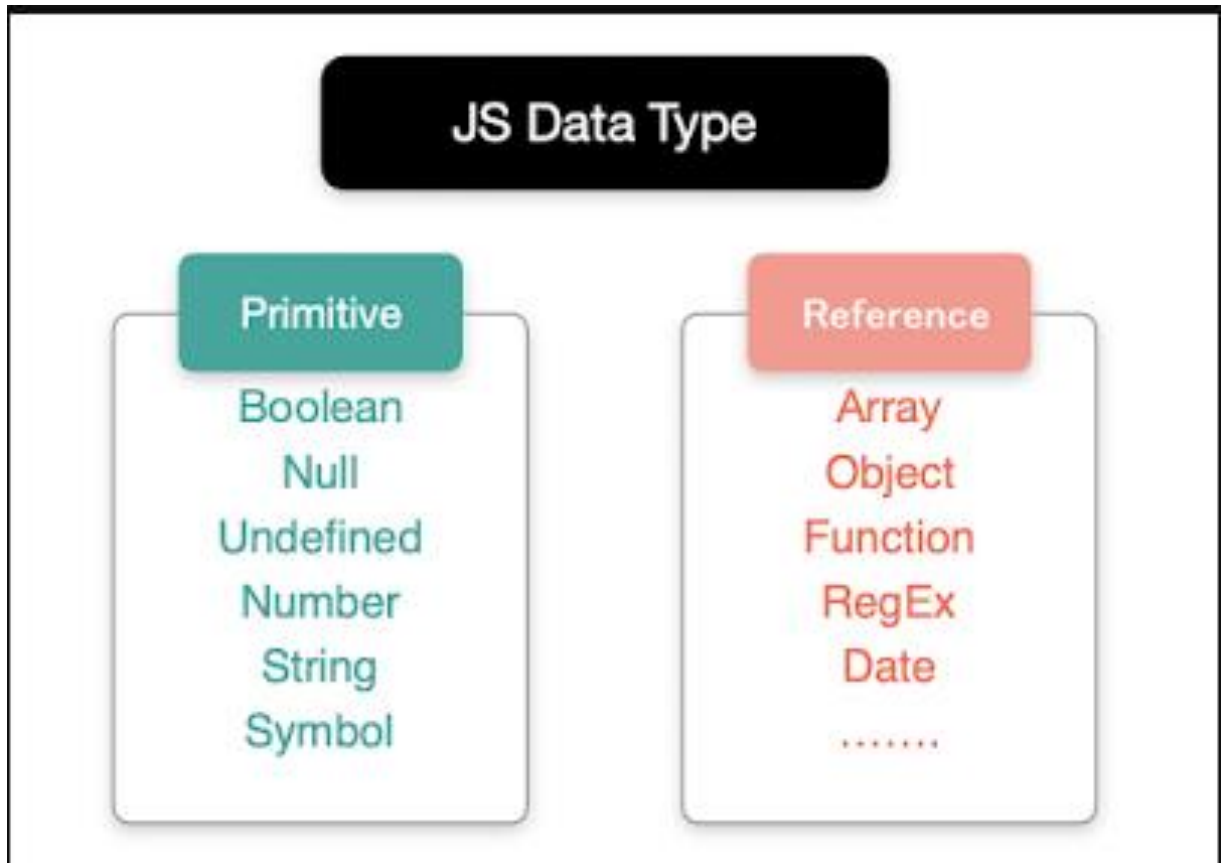
Other Reserved Words to Avoid

Here is a list of words to avoid that are are used as names of properties, methods and consructors in JavaScript. You should also try not to use these names as you risk overwriting existing functionality in JavaScript:

alert	closed	innerWidth	Object	prompt	statusbar
arguments	confirm	isFinite	open	prototype	stop
Array	constructor	isNaN	opener	RegExp	String
blur	Date	java	outerHeight	releaseEvents	toolbar
Boolean	defaultStatus	length	outerWidth	resizeBy	top
callee	document	location	Packages	resizeTo	toString
caller	escape	locationbar	pageXOffset	routeEvent	unescape
captureEvents	eval	Math	pageYOffset	scroll	unwatch
clearInterval	find	menubar	parent	scrollbars	valueOf
clearTimeout	focus	moveBy	parseFloat	scrollBy	watch
close	frames	name	parseInt	scrollTo	window
home	Function	NaN	personalbar	self	setTimeout
Infinity	history	netscape	print	setInterval	status
innerHeight		Number			

JS variables and types

JavaScript provides different **data types** to hold different types of values. There are two types of data types in JavaScript.



1) Primitive data type
2) Non-primitive (reference) data type

Flow Control and Looping

Conditional Statements

if statements

if ... else ... Statements

Looping Statements

while loops

do ... while loops

switch Statements

label Statements

Continue Statements

Functions, Arrow Functions

- Function declarations
  Function expressions
- Calling functions
- Function hoisting
- Function scope
- Nested functions and closures

```
function addSquares(a, b) {
        function square(x)  {
            return x * x;
        }
        return square(a) + square(b);
    }
```

- Function parameters
- Arrow functions

JavaScript Array is a single variable that is used to store elements of different data types. JavaScript arrays are zero-indexed. The Javascript Arrays are not associative in nature.
Arrays are used when we have a list of items. An array allows you to store several values with the same name and access them by using their index number.

JavaScript object is a non-primitive data-type that allows you to store multiple collections of data.

JavaScript Scope

A scope can be defined as the region of the execution, a region where the expressions and values can be referenced.

There are two scopes in JavaScript that are global and local:

Global Scope: In the global scope, the variable can be accessed from any part of the JavaScript code.
Local Scope: In the local scope, the variable can be accessed within a function where it is declared.

- What is Block Scope?
- What is Function Scope?
- JavaScript Strict Mode for Defining Scope of a Variable

Hoisting

JavaScript **Hoisting** refers to the process whereby the interpreter appears to move the *declaration* of functions, variables, classes, or imports to the top of their scope, prior to execution of the code.

When the JavaScript engine executes the JavaScript code, it creates the global execution context. The global execution context has two phases:

1. Creation
2. Execution

During the creation phase, the JavaScript engine moves the variable and function declarations to the top of your code.

- ❑ Variable hoisting with let and const
- ❑ The temporal dead zone
- ❑ typeof in the temporal dead zone

Execution Context

Execution Context Two Type

1) Global Execution Context(GEC)
2) Function Exaction Context (FEC)

The creation of an Execution Context (GEC or FEC) happens in two phases:

a) Creation Phase

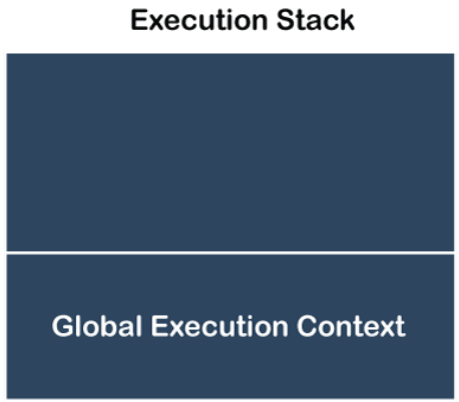
The creation phase occurs in 3 stages, during which the properties of the Execution Context Object are defined and set. These stages are:

- ➔ Creation of the Variable Object (VO)
- ➔ Creation of the Scope Chain
- ➔ setting the value of the this keyword

b) Execution Phase

The Variable Object (VO) is an object-like container created within an Execution Context. It stores the variables and function declarations defined within that Execution Context.

Execution context stack (ECS)



Execution Context

Let's start with the following example:

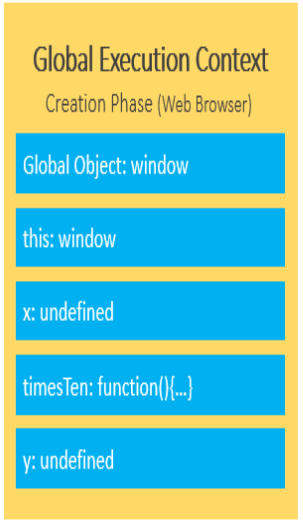
```
let x = 10;

function timesTen(a){
    return a * 10;
}

let y = timesTen(x);

console.log(y); // 100
```

In this example:

- First, declare the `x` variable and initialize its value with `10`.
- Second, declare the `timesTen()` function that accepts an argument and returns a value that is the result of multiplication of the argument with `10`.
- Third, call the `timesTen()` function with the argument as the value of the `x` variable and store result in the variable `y`.
- Finally, output the variable `y` to the Console.

When the JavaScript engine executes the code example above, it does the following in the creation phase:

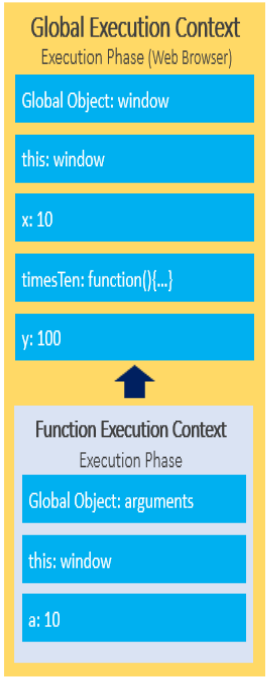
- First, store the variables `x` and `y` and function declaration `timesTen()` in the global execution context.
- Second, initialize the variables `x` and `y` to `undefined`.



After the creation phase, the global execution context moves to the execution phase.

In our example, the function execution context creates the `arguments` object that references all parameters passed into the function, sets `this` value to the global object, and initializes the `a` parameter to `undefined`.

During the execution phase of the function execution context, the JavaScript engine assigns `10` to the parameter `a` and returns the result (`100`) to the global execution context:



Polyfill

A polyfill is a piece of code (usually JavaScript on the Web) used to provide modern functionality on older browsers that do not natively support it.

Polyfill Considerations:

While polyfills are incredibly useful, it's important to consider a few points when using them

- Selective Loading: Polyfills should only be loaded for browsers that need them. Modern bundlers and package managers often provide ways to conditionally load polyfills based on browser support, helping optimize the performance of your application.

- Feature Detection: Before applying a polyfill, it's crucial to perform feature detection to avoid overwriting native implementations or applying unnecessary polyfills.

- Size and Performance: Polyfills can increase the size of your JavaScript bundle. Minification and compression techniques can help mitigate this issue, but it's essential to balance the benefit of polyfills against their impact on performance.

- Using Existing Libraries: Rather than reinventing the wheel, it's worth exploring existing libraries and tools that offer comprehensive polyfill solutions for multiple features. These libraries often handle feature detection, conditional loading, and fallbacks efficiently.

This keyword in JS

What is **this**?

In JavaScript, the this keyword refers to an **object**.

Rules of this keyword

In an object method, this refers to the **object**.
Alone, this refers to the **global object**.
In a function, this refers to the **global object**.
In a function, in strict mode, this is undefined.
In an event, this refers to the **element** that received the event.
Methods like call(), apply(), and bind() can refer this to **any object**.

Custom Rules

1) Global
2) Object
3) variable
4) new keyword