**JAVASCRIPT NOTES:**

=> PRIMITIVE DATA TYPES:

1. NUMBER
2. STRING
3. BOOLEAN
4. UNDEFINED
5. NULL
6. SYMBOL { USED FORT A UNIQUE VALUE WHICH DO NOT CHNAGE}. ES2015
7. BIGINT. { LARGER VALUES THAN THE NUMBER DATATYPE CAN HOLD } ES2020

=> TYPES OF FALSY VALUE IN JAVASCRIPT:

1. 0
2. ‘ ’ i.e empty string
3. NaN
4. Null
5. Undefined

=> TYPES OF EQUALITY OPERATOR

1. STRICT EQUALITY OPERATOR ‘===, [ DO NOT PERFORM TYPE COERCION]
2. LOOSE EQUALITY OPERATOR ‘==‘. [ PERFORMS TYPE COERCION]

EXAMPLE:

’18’==18 gives true.

Why? [[ performs type coercion and thus ignores the dataypes]]

’18’===18. gives false.

Why? [[ strictly checks the datatype also]]

=> \*\* IMPORTANT \*\* TYPES OF FUNCTION::

FUNCTION DECLARATION:

——————————————

Example:

function presentAge(year)

{

return year-1991;

}

const result= presentAge(2022)

Console.log(resulr)

FUNCTION EXPRESSION

—————————————

Example:

Const age = function(year). //a function with no name I.e anonymous function is stored inside a variable

{

return year -1991;

}

const result = age(2022) ;

console.log(result);

DIFFERENCE BETWEEN FUNCTION DECLARATION AND FUNCTION EXPRESSION

———————————————————————————————————————————

We can invoke function declaration even before they are defined in the code whereas we cannot invoke function expression before they are defined.(related to hoisting)

ARROW FUNCTION. (ES6)

———————————

Basic Syntax : params => expression(value to be returned). Here we don t use the keyword return

Example: const age = birthYear =>. 2022 - birthYear;

OBJECTS :

—————

EXAMPLE OF OBJECT:

const obj = {

First name:’Shanky’,

Age: 22,

Profession:’IT’,

Salary: 100000 -20000

}

Dot v/s Bracket Notation:

—————————————

Example of Dot Notation: obj.firstName

Example of Bracket Notation: obj [ ‘ firstName ’ ] //here we can compute the values also

Object Methods:

—————————

This keyword

———————-

this keyword when used in a function which is inside an object then it gives reference to that object only.

but when we use this keyword with a regular function then it references to the global execution I.e window object,

But if we call the same function with the help of new keyword it creates a new object and references to .

JAVASCRIPT IS A MULTI PARADIGM LANGUAGE:

TYEPS OF PARADIGM

1. PROCEDURAL
2. OBJECT ORIENTED
3. FUNCTIONAL PROGRAMMING

WHAT IS FIRST CLASS FUNCTION? \*\*\*\*(IMPORTANT)\*\*\*\*\*

—> IN JAVASCRIPT FUNCTIONS ARE TREATED AS VARIABLES . WE CAN PASS THEM INTO ANOTHER FUNCTION AND ALSO RETURN THEM FROM A FUNCTION

\*\*\*IMPORTANT\*\*

JAVASCRIPT IS SINGLE THREADED LANGUAGE I.E PERFORMING ONE TASK AT A TIME

HERE COMES IN GAME THE CONCURRENCY MODEL I.E HOW TO HANDLE DIFFERENT TASK RUNNING AT THE SAME TIME.

WHAT ABOUT LONG RUNNING TASKS IT WOULD BE BLOCKING THE SINGLE THREAD SO HER COMES THE CONCEPT OF “EVENT LOOPS” WHICH BASICALLY TAKE THE LONG RUNNING TASKS -> EXECUTES THEM IN BACKGROUND -> AGAIN PUT THEM BACK IN THE MAIN THREAD.

AN DTHIS MAINTAINS THE NON BLOCKING BEHAVIOUR OF JAVASCRIPT.

SCOPING AND SCOPE CHAIN

————————————————

SCOPING IS BASICALLY HOW ARE PROGRAMS VARIABLE PLACED AND ACCESSED.

SCOPING IS RELATED TO THE TERM LEXICAL ENVIRONMENT (I.E SCOPIG WHICH IS CONTROLLED BY PLACING OF FUNCTIONS AND BLOCKS IN THE CODE)

THERE ARE THREE TYPES OF SCOPE:

1. GLOBAL SCOPE
2. FUNCTION SCOPE (ALSO KNOWN AS LOCAL SCOPE)
3. BLOCK SCOPE (ANYTHING WHICH IS BETWEEN CURLY BRACES SUCH AS IF ELSE AND FOR LOOPS)

ONLY LET AND CONST ARE THE ONES ON WHICH THIS BLOCK SCOPING APPLIES AND HERE WE COME ACROSS THE FAMOUS STATEMENT

“LET AND CONST ARE BLOCKED SCOPE”

AND EVEN IF WE USE VAR INSIDE A BLOCK IT WILL END UP TO BE IN THE CLOSEST FUNCTION .

WHAT IS HOISTING ?

———————————

USING VARIABLES EVEN BEFORE THEY ARE ACTUALLY DECLARED

CONSOLE.LOG(A);

LET A=50;

OUTPUT: UNDEFINED

HOISTING BEHAVIOUR FOR DIFFERENT THINGS:

—————————————————————————

1. FUNCTION DECLARATIONS HOISTED VALUE == ACTUAL/ENTIRE FUNCTION
2. VAR. HOISTED VALUE == UNDEFINED
3. LET AND CONST. NOT HOISTED VALUE== GIVES REFERENCE ERROR (TDZ I.E TEMPORAL DEAD ZONE)
4. FUNCTION EXPRESSION. HOISTED. VALUE==UNDEFINED (THEY ACT AS A VARIABLE). AND SECONDLY DEPENDS UPON THE TYPE OF VARIABLE EITHER VAR OR LET/CONST IS USED

WHY WE USE HOISTING IN JS?

————————————————

TO USE FUNCTION BEFORE ACTUAL DECLARATION

\*\*\*\*\* V.V Important: \*\*\*\*\*

this Keyword

——————

This keyword is created for every execution context(i.e every function)

Takes the value(point to) the owner of the function in which this keyword is used.

‘this’ is not static. It depends on how the function is called

Its value is assigned when the actual function is called

Method: this will reference to the object itself (this keyword will point to the object which is calling the function)

Simple Function: in strict mode it this == undefined whereas without strict mode its will refer to the window/global object

Arrow Function: They do not get their own this keyword so if you this keyword with arrow functions it will give reference

to the parent function (we can see it as lexical this) or we can say global scope

Event Listener : this will give reference to the DOM element to which the handler is attached

\*\*\*\*\*IMPORTANT\*\*\*\*\*

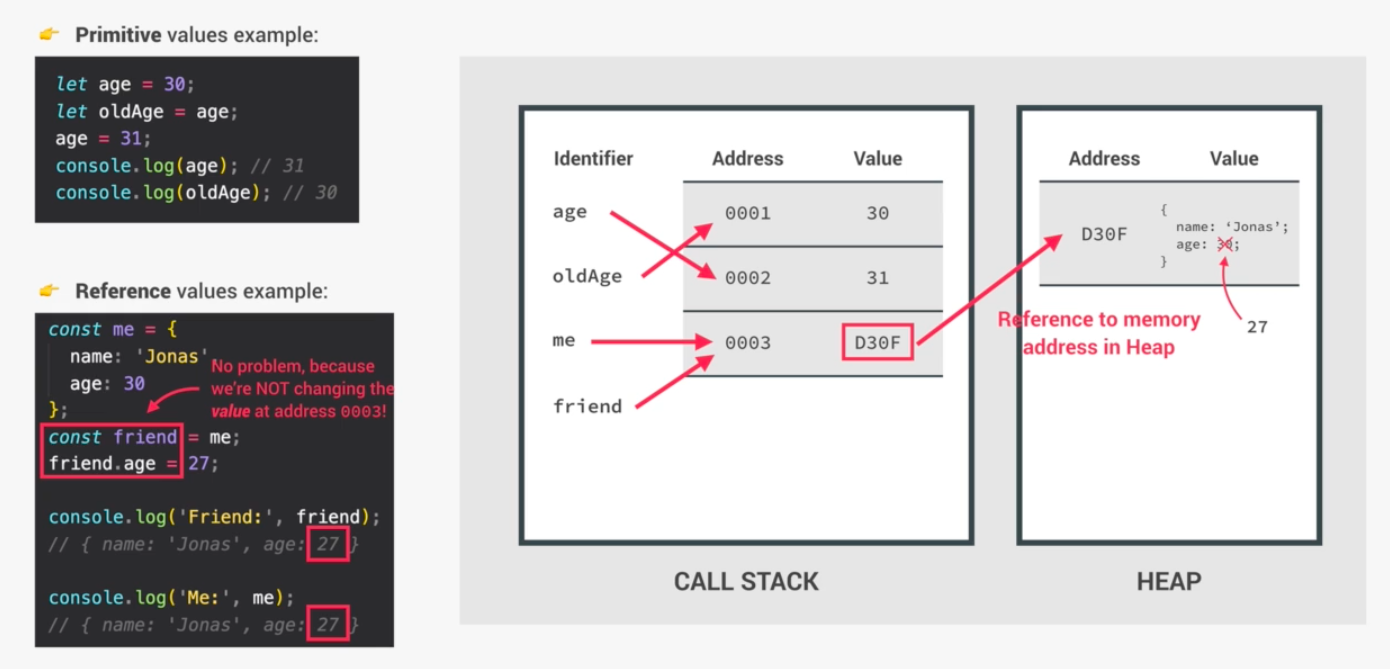
Main Point: “this” will NEVER point to the function itself and It won’t point to the VARIABLE ENVIRONMENT

PRIMITIVE V/S REFERENCE VALUE

——————————————————

THE PRIMITIVE DATATYPES SUCH AS NUMBER, STRING, BOOLEAN, NULL, UNDEFINED, SYMBOL AND BIGINT ARE STORED IN CALL STACK , WHEREAS

OTHER SUCH AS OBJECTS ,FUNCTIONS ETC ARE STORED IN THE HEAP MEMORY.



SO IF WE ARE COPYING AN OBJECT INTO OTHER AS GIVEN IN THE ABOVE EXAMPLE WE WILL FACE A PROBLEM THAT IF WE WANT TO MAKE CHANGES IN THE COPIED OBJECT THEN IT WILL REFLECT THE CHANGES IN THE ORIGINAL OBJECT TOO AS IT THE COPIED OBJECT ONLY GIVE REFERENCE TO THE ORIGINAL OBJECT

SO IN ORDER TO OVERCOME THIS PROBLEM WE USE:

OBJECT.ASSIGN ( SOURCE ,TARGET ). //THIS WILL CREATE A NEW OBJECT IN THE HEAP WITHOUT GIVING REFERENCE TO THE ORIGINAL OBJECT AND

HENCE CHANGES WILL BE DONE ONLY TO THE COPIED ARRAY.

CLOSURES:

