Javascript

Strict Javascript

Strict Mode.

‘use strict’;

Requirements.

-Install node.js
-npm install live-server -g

1.Variable
-Variable Declarations.
-Changing Variable Value.
-Let Const var.

2.Data Types.
-String.
-Numbers.
-Boolean.
-Undefined.
-Null.

3.Template Literal
`{yourName}`

4.Type Conversion/Coercion
-Converting String to Number. Number().
-Converting Number to String. String().
- “12”-2+”12” = 1012

3.Operators
-Arithmetic operators.
-Assignment.
-Comparison operators.

-Logical Operators.

4.Conditional Statements.
-if
-else
-else if
-Switch

5.Ternary Operator

?
condition? Expression if truth : expression if false
6.Functions
-Declaring Functions.
-Passing Parameter And Arguments.
-Declaring variable when returning Arguments and logging it.
-Function Declaration vs Expression
-Arrow Function => or ()=>
-Using different functions together - Argument Points.

7.Arrays
-Ways to write Array [] , new Array().
-Changing Array Values.
-Different data types in Array.
-Array Methods:
    length()
    pop(),push()
    Shift(),unshift(),
    splice()
    split()
    replace()
    indexOf() 0,-1 includes() true0r false

8.Object
-What an Object is.
-Syntax of Object(Object literal)
-Different Data Types in Objects.

-Accessing a certain property in an Object.
-Dot and Bracket Notation.
-This keyword

9.Loops

-For loop.
-Loop Syntax -> for(initialization of condition; testing condition; increment/decrement){ statement }
-For loop in arrays.
-Reading Arrays
-Creating new based on the value of one array. Using push.
-Continue and Break Statements.
-Looping Backwards in an Array.
-Nested Loops.
-While Loop.
10.Math Object
-What is it?
-Math Methods.
-Math Properties
-Methods Used in math.

11.When to Use function expressions,declarations,Arrow functions.
-Functions Expressions.
-Function Declarations.
-Arrow Functions.

12.Scoping
-What is it
-Global Scope
-Local Scope

13.Hoisting
-What is it
-Using a Variable.
-Using function expression

-Using function declarations.
-Using Arrow Functions.

12.Primitives vs Objects

13.Destructuring Arrays && Objects
-New way vs Old way.
-Destructuring Syntax in Arrays. []
-Destructuring Syntax in Objects {}
-Spread Operator.
-Spread Operator in Arrays.
-Rest Operator.
-Rest Operator in Functions.

14.Short Circuiting
-What is it?
-Truthy and Falsy Values
-Its Operators And,or and Not. &&.||.!
-Nullish operator

15.For of Loop
-In array
- .entries in arrays

16.Enhanced Object Literals
-Old Way
-New Way

17.Arrays,Object,Functions
-Using Arrays index to as Object Keys.
-Declaring functions without assigning key names to objects.

18.Optional Chaining operator ?.
-Defination.
-Using it with Nullish operator

19.Looping over Object
-For of loop is Used here.
-Object.keys() function
-Object.values() function
-Object.Entries() function:(Return Object keys and values as arrays,Hence Destructuring Here is possible)

20.Set and Map
-Set Definition and syntax.
-Set Methods.
-Main function of set.
-Map Definition and 2 types of syntax.
-Map Methods.
-Looping through Maps.

21.String Methods
-Length
-indexOf()
-lastIndexOf()
-slice()
-replace()
-includes()
-startsWith()
-endsWith()
-toUpperCase()
-toLowerCase()
-trim()
-split()
-join()
-repeat()

22.Default Parameter
-What is it?
-Old vs New way of setting default parameters.

23.First Class and Higher Function
-What are They?
-Function as 1st class citizen:-
-Variable assignment to function.
-Returning a function from a function.
-Passing a function as an argument in another function.

24.Call and Apply Method
-Defination:- They used to call methods from various objects that had this keyword.
-You can use one method to write different objects,Key names are same.
-Call and Apply syntax

```
call(objectName,  function arguments goes here)
apply(objectName, [ function arguments goes here])
```

25.Bind Method
-Defination.
-Syntax

Variables

-It is used as a storage container for values.
-Variables are declared using “var”,let,const.

Variable Declarations

1.Camel Case notation-var firstName
2.Dollar notation- var $firstName
3.Do not start with Uppercase, With uppercase its reserved for constant values.
4.Avoid using javascript like new,name.

```
var firstName = “Koech”;
var first_Name = “Koech”;
var $firstName = “Koech”;
var firstName = “Koech”;
```

Changing our variable value.

-Here we don't have to declare the variable again.

-Instead we just call the variable name and declare the new value.

-Then console.log to see your new value.

```
var firstName = “Koech”;
firstName = “Emmanuel”;
console.log(firstName)
```

Let,Const,Var

-Let is used to declare variables where its values that will change later,An example is our age changes as we grow.

```
let age = 32;
age = 33;
console.log(age)
```

-Const is used to declare variables that cannot be changed,,An example is our birth year never changes.

```
const birthYear = 2002;
birthYear = 2003;
console.log(birthYear)
```

- An error will occur upon execution.

NB: -It is a good practice to use let or const when writing clean code.Use const in instances where the variable values are immutable(Cannot change) else where the variable values are mutable(Can change) use let.

Data Types

Primitive Data Types

1.Number:- Used for decimals and integers.

```
var age = 12;
```

2.Strings:-Used for text.

```
var name = “emmanuel” ;
```

3.Booleans:-Used to make decisions, either true or false.

```
Var fullAge = true;
```

4.Undefined:-This is when the value taken by value is not yet defined.

```
var myName ;
```

5.Null:- This means empty value.

```
Var myName=;
```

6.Symbol

7.BigInt:-Larger integers than Number can hold

NB:- Javascript has dynamic typing.We don't have to manually define datatype of the value stored in variable instead it is determined automatically.

Type of

-It checks the type of datatype.

```
var firstName = “Emmanuel”;
console.log(typeof firstName)
```

Operators

-They are used to perform mathematical operations,compare values.

- Arithmetic Operators

- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

1.Arithmetic Operators

-They are used to perform arithmetic operations

+ Addition Operator
- Subtraction Operator
* Multiplication Operator
/ Division Operator
** Explanation operator
++ Increment Operator
– Decrement Operator
% Modulus Operator (Remainder)

2.Assignment Operators

= x=y
+= x=x+y
-= x=x-y
*= x=x*y
/= x=x/y

3.Comparison Operators

=== - Equal to
!== - Not Equal to
> - Greater than
< - Less than
>= -Greater than or equal
<= -Less than or equal to

? -Ternary Operator
==! -Not equal value or type
=== - equal value equal type

4.Logical Operator

&& -And
|| -Or
! -Not

Logical Operators

The And operator takes condition that is only True while or takes condition that can either weight True or False.

```
Const hasDriverLicense = True;
Const hasGoodVision = True;

if(hasDriverLicense&&hasGoodVision){
                console.log(“You allowed to Drive”)
                                }else{
                        “You are not allowed to drive}
```

-Here the driver will be allowed to driver cause he

```
Const hasAttainedLegalAge = True;
Const hasVerifiedID = True;

if(hasAttainedLegalAge || hasVerfiedID){
                console.log(“You allowed to pass”)
                                }
```

Template Literals

Template literals use backtick(``) instead of quotes(“”).

let myName = `I am Emmanuel`; (Backtik)
let myName = “I am Emmanuel”; (Quotes)

-With template literal you can use both quotes and backtik.
Let myName = `They often call me “Manu”`;

-Template literal allow multi string.
Let myName = `They often
call me
“Manu”`;

-Template literal allows string interpolation.
let firstName = "John";
let lastName = "Doe";
let text = `Welcome ${firstName}, ${lastName}!`;

if ,else

-Conditional statements are used to perform different actions based on different conditions.

The syntax:

if(condition){

If the block of code to be executed is true.

}else if(condition){

If the block of code to be executed is true.

}else{

If a block of code to be executed is false.

}

Conditional Statements in Javascript

If

Else

Else if

Switch

Switch

-It is used to perform different actions based on different conditions.

-The Switch statement is used to select one of the many codes to be executed.

-The default keyword is used to specify the code which will run if there is no case match.

-The case represents the value of variable declare.For the case to repeated again we call it then break it on the second case.

NB Like this:-

```
switch(){
    Case:
    Case:
        console.log()
        Break;
        Default:
        console.log()


        }
```

Syntax

```
switch(expression){
    case a:
    code block
    Break;
    case b:
    code block
    Break;
    Case c;
```

```
        case d:
        code block
        Break;
        default:
        Code block
}
let dayOfWeek = `Monday`
switch(dayOfWeek){
                case Monday:
                console.log(“This is a tough work day”)
                Break;
                case `Tuesday`:
                console.log(‘This is a less tough day compared to Monday’)
                Break;
```

```
        case Wednesday:

        console.log(`This a average day,Champions League Night makes it
    a day for me`)

        Break;

        Case Thursday:

        Case Friday:

        console.log(`Wow This is so fun,Sherehe Day`)}

        Case Saturday:

        Case Sunday:

        console.log(`You Have to go to Church during this`)

        break

        default:

        console.log(`Invalid Day `)

        }
```

Ternary Operator

-This is the only javascript operator that takes three operands.

-A condition followed by ?.

-An expression to execute if condition is truthy followed by colon:

-Then the expression to executed if condition is falsy

Sample:
Let age = 15;

age>18? console.log(`You an Adult`) : console.log(`You a Child`)

Syntax

condition ? expressionIfTrue : expressionIfFalse

Same as:

Let age = 18;

if (age>18){

console.log(`You an adult`)

}else{

console.log(`You a child`)

-Here is where we use the tertiary operator in real situation.

-It Shortens our code and makes it more readable.

-We defined our variable drink then took our age as condition.

-Later the block runs checking the specific condition

Let age = 18;

Let drink = age<18 ? `water` : 'Any Drink' ;

let waiter = `I am bringing you ${drink}`;

```
console.log(waiter)

 }
```

Type conversion / Type Coersion

Type conversion is the process of converting data of different types to other types.

Converting String to Numbers. Number()

```
let yearOfBirth = “2002”
console.log(number(yearOfBirth))
Or
let yearOfBirth = “2002” ;
yearOfBirth = number(yearOfBirth)
console.log(yearOfBirth)
```

Converting Numbers to String. String()

```
let yearOfBirth = 2002
console.log(string(yearOfBirth))

Or

let yearOfBirth = 2002;
yearOfBirth = string(yearOfBirth)
console.log(yearOfBirth)
```

Type of coersion

```
let myAge = 13
 myAge = “13”-2+”13”
console.log(myAge)
```

The sum here will be 1113.
This is because we can subtract an integer string “13” from a number 2.
In the process of addition the integer is added to the string,Which cannot be added..

Truthy And Falsy Values

Truthy value are values that translate to true when evaluated to boolean.
All values are Truthy unless they evaluated they defined falsy.

-Null
-NaoN
-Undefined
-0

Functions
-Functions are used to perform actions and allow us to write more maintainable code
-They can be reused in many instances.

How to write functions
1.Define the function with the function keyword function which has parenthesis ().
2.Name your function, function nameOfFunction() this is followed by curly brackets {}.
3.Call function,Invoke Function,Running Function.

Syntax

```
function name_of_function(){
                        Code to be executed goes here
                                }
        name_of_function()
```

console.log

```
function name_of_function(){
                    console.log(“Data”)
                                }
        name_of_function()
```

Return

```
function name_of_function(){
                    return “Data”
                                }
 var input = name_of_function()
console.log(input)
```

Parameters & Arguments

```
function name_of_function(first_name,second_name){

                    return `Hello ${first_name} ${second_name}`
                                            }
Var your_Name = name_of_function(“Manu”,”Koech”)
console.log(your_Name)
```

Function Declarations vs Function Arguments.

Function declaration.

```
function yourAge(birthYear){
                return 2022 - birthYear;
                    }
let personAge = yourAge(2002);

console.log(person)
```

Function Expression

```
Let yourAge = function(birthYear){
                return 2022-birthYear
               }
```

yourAge(2002)

-With function expression we declare our function in a variable.
-In our values we declare our function but without the name as we do.
-When we call our function we call it with the variable name

Arrow Functions

-Arrow Functions allows us to write shorter Function Syntax.
-In arrow function our function name is not declared.
-Also we don't have to mention the return.
-If you want to put one parameter it comes before the arrow function.
-If we declare 2 parameters then we put calibraces () followed by an arrow function.
-Also Arrow Function does Not Accept this keyword as the result will be undefined.

NB -Never Use an Arrow function as a method.

```
let yourAge =  birth year => 2022 - birth year;
let yourCurrentAge = yourAge(2002)

console.log(greet("Emmanuel))
console.log(yourCurrentAge)
```

-Note if you have to pass 2 parameters we add ()=>

```
let yourAge = (birth year,currentYear)=>currentYear - birth year
let youCurrentAge= yourAge(yourAge(2002,2022))
```

Using different Functions together.

-Since function are used to perform action, they can be used to perform different actions to our code which are functions too..
-We create our function which is suppose to perform a specific action `calculate age`.
-Afterwards we create another function which checks the person who is older.
-This code uses the first function ‘calculate age’ to check who is older.
-It uses the arguments to fetch the variable of the people whose age were calculated

Here is a function that calculates our Age

```
let calcAge = (birthyear,currentYear)=> currentYear -birthyear;
let ageManu = calcAge(2002,2022)
let ageSharon = calcAge(1990,2022)
```

The function which checks who is older.

```
let olderPerson = (personOne,personTwo){
                    if (personOne>personTwo){
                    return `You are  {personTwo-personOne} older`}else{
                    return `You are {person} younger`}
Let personResults = olderPerson(ageManu,ageSharon)
```

Once again The Arguments in this function take the variable of people whose age were calculated using the calc Functions.

Arrays

-This is a list of items,It stores more than one value.
-Instead of storing one value in our variable we can create an array.
-An array is created with [] or new Array().The values go into the brackets.
-Arrays can hold different data types also.

```
let myFriends = ['Ian',`Sharon`,`Lorem`];
let myFriends = new Array('Ian',`Sharon`,`Lorem`);
```

Replacing array values

-We can replace the values of our array by calling the variable with its index value.

```
let myFriends = ['Ian',`Sharon`,`Lorem`];
 myFriends[0]=`Sharon`;
console.log(myFriend)
```

Array Methods

length()

-Checks the length of Array

```
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
        console.log(items.length)
```

push()

-Adds element at end of array.

```
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.push(“buffalo”)
console.log(items)
```

Buffalo will be added in the array

Pop()

-Removes last element of an array and also prints it in terminal.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.pop()
console.log(items)
```

Hen will be removed from the array

shift()

-Removes first element of an array.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.shift()
console.log(items)
```

cows will be removed from the array

unshift()

-Adds first element of an array but overwrittes it.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.unshift(“buffalo”)
console.log(items)
```

buffalo will be added from the array

splice()

-Its is used to remove data in array or add data from a certain index number.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.splice(2)
```

-Here all items from index 2 will be removed.

```
items.splice(2,3)
```

-Here all items from index 2 to 3 will be removed the rest remains

```
items.splice(2,3 11, 12)
```

-Here all items from index 2 to 3 will be removed the rest remains but 11,12 will be added in the array as we said splice is used to add data in arrays also.

indexOf()

-It returns the indexOf the value specified from the array.

-If the value specified is not found it returns -1

-It searches from left to right.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`cows`))
```

0 will be the return value at the terminal.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`buffalo`))
```

-1 will be the return value at the terminal since there is no such value in our array.

Includes()

-Returns either True or False if the value is in the array.

```
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.include(`cows`))
```

True will be the return value at the terminal since cow is present in the array.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`buffalo`))
False will be the return value at the terminal since there is no such value in our array

Split()
-Used to Split a string into array of items.

Slice()
-Used to Remove letters from a string

Methods

JavaScript methods are actions that can be performed on objects.

```
var laptops ={
                brand:”HP”,
                cpu:”4GB”,
                disk_drive:”SSD”,

                greet: function(){
                        return “Hello World”
                        console.log(“Hello Manu”)
                        }
```

```
}
var results = laptops.greet()
console.log(results)
```

This

-It is used to refer to a certain object.

Here are my objects.

```
var car = {
                origin : “Japan”,
                module : 2010,
                engine : “Diesel”,
                myCar : ()=>{
                                console.log(this.origin)
                                        }
                    car.myCar()

var car2 = {
                origin: “Singepore”,
                module : 2010,
                engine : “Diesel”,

                    myCar : ()=>{
                                                console.log(this.origin)
                                        }
                    car2.myCar()
```

Output
Car1 = Japan
car2= Singapore

Constructor Function

-Instead of creating many object with the same blueprint as we have done above we can use a constructor function to help us avoid repeating ourselves.

-Function name starts with a capital letter.
-This keyword is used to refer to a certain object.
-When called we use new then name of function,We get the new instance of our object.
-The “new keyword” creates a new empty object,then assigns the argument to the properties of object

```
function Car(origin,module,engine){
                this.origin=origin,
                this.module=module,
                this.engine=engine
                 }

var Car1= new Car(“Japan”,2010,”Diesel”)
var Car2= new Car(“China”,2020,”Diesel”)

console.log(car1,car2)
```

Arrays

-This is a list of items.

Creating arrays

var cars = [“Toyota”,”Benz”,”Bently”]

Creating different data in arrays

var myNeeds = [“towel”,{car:gwagon},2<3,5,function(){console.log(“Manu”)}]
console.log(myNeeds)

-Here we have different data in our array and we logged it at the end.
A function is also included.

Adding a new value we use push():

cars.push(“Gwagon”)

Finding length of array we use length():-

cars.length()

Fetching array

console.log(cars.length)

Methods in Array

push() -Adds element at end of array.

```
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.push(“buffalo”)
console.log(items)
```

Buffalo will be added in the array

Pop() -Removes the last element of an array and also prints it in terminal.

```
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.pop()
```

console.log(items)
Hen will be removed from the array

shift() -Removes the first element of an array.
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.shift()
console.log(items)
cowa will be removed from the array

unshift() -Adds first element of an array but overwritten it.
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.unshift(“buffalo”)
console.log(items)
buffalo will be added from the array

splice()-Its is used to remove data in an array or add data from a certain index number.
Var items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.splice(2)
-Here all items from index 2 will be removed.
items.splice(2,3)
-Here all items from index 2 to 3 will be removed the rest remains
items.splice(2,3 11, 12)
-Here all items from index 2 to 3 will be removed the rest remains but 11,12 will be added in the array as we said splice is used to add data in arrays also.

Object

-A javascript object is a collection of named values.

Here is a javascript object:-
We have used the object literal here.

Syntax:
const variableName = {

```
Key:value,
Key:value,
Key:value,
}
```

```
Const person = {
    firstName:`John`,
    lastName:`Doe`,
    Age:50,
    eyeColor:`blue`,
        };
```

-Here our key is the firstName and the rest.

-The value are the names taken after the key.

Fetching Specific values from Objects

-This is where we fetch the value of our choice from the Object.

-We can use the dot notation.

-We can use the bracket notation,but the key will be put in string.

-Here we call our Variable plus the key in our object.

-Our Output will be the value of the Key

```
Const person = {
    firstName:`John`,
    lastName:`Doe`,
    Age:50,
    eyeColor:`blue`
        };
console.log(person.firstName)
console.log(person[`firstName`])
```

Bracket Notation

-With bracket notation we can concatinate our key with a new variable.

-We can use an existing property key with a new variable as shown below.
-Our Output will still be the value of the key in our Object.

```
const person = {
        firstName:`John`,
        lastName:`Doe`,
        Age:50,
        eyeColor:`blue`
            };

Const nameKey = `Name`
console.log(person[`first`]+nameKey)
```

Adding New Object Properties

-We simply call the variable name of the object plus the new key value.
-Below property will be our new property.

```
Const person = {
        firstName:`John`,
        lastName:`Doe`,
        Age:50,
        eyeColor:`blue`
            };
```

Using Dot Notation

```
person.height = 10;
```

Using Bracket Notation

```
Person[`age`] = 11;
```

Functions in Objects

-Function expression can be stored in Objects
-We declare our function in the key instead of variable then the value is the function.

```
const person = {
        firstName:`John`,
        lastName:`Doe`,
        Age:50,
        eyeColor:`blue`
        yourAge : birthYear => return 2022 - birthYear,
            };
console.log(person.yourAge(2002))
```

This Keyword
-It is Used to refer to a certain Object.
-For how I understand it helps us maintain DRY,Dont Repeat yourself.
-When used alone it outputs the global object.

```
const person = {
        firstName:`John`,
        lastName:`Doe`,
         currentYear:2022
        birthyear : 2002;
        Age:50,
        eyeColor:`blue`
        yourAge : birthYear => return this.currentYear - this.birthYear,
            };
console.log(person.yourAge(2002))
```

Loops

-Loops are used to repeat a block of code.

For loop

-For loop, loops through a code a number of times.
-It keeps running if condition is true.
-It has 3 expressions which is as follows in the syntax.

for(initialization of condition; testing condition; increment/decrement){
statement }

Initialization of condition.

-We initialize the loop with a variable and value which is an integer ,and this marks where our loop starts.

Testing Condition.

-Here is the the stopping condition, It determine when the loop has to stop.
-Here our condition is tested and if it is true our loop body is executed.

Increment/Decrement.

-This increases the counter otherwise our code will remain at 1 forever.

An example

```
for(let i = 0 ; i<10 1++){
                        console.log(`Hello there${i}`)
                        }
```

-Here we have our loop which starts at 0.
-It condition is set to less than 10,Where it will stop.
-An increment of i=i+1 is set which mean it will increase by one.
-In our console we concatinate with the variable so as to see the sequence.

Looping in Arrays.

-We can loop in a list of values or items which is an array.
-Instead of logging the array items one by one by using its index number A loop can be used to print them one by one.
-We first initialize our loop with a variable whose value is 0 because arrays start from 0.
-Our condition is the length of array ,which is where our loop stops, we use the length function.
-We log our array with the counter

```
Const jonas = [`Jonas`,`Schmedinarm`,2037-1931,`Teacher`,true]
for(let i = 0; i<jonas.length;i++){
                        console.log(jonas[i])
                        }
```

Above we have been Reading an Array.

Creating Arrays based on the value of one original Array.

-Create an empty array outside the loop.
-Inside our loop we use the push method to add our original array.
-When we log our empty array outside the loop it should return the type of data similar to our original array.

```
let jonas = [`Jonas`,`Schmedinarm`,2037-1931,`Teacher`,true];
let type = [];

for(let i = 0;i<jonas.length;i++){
        type.push(typeof jonas[i])
                                }

console.log(type)
```

The Datatypes will be the output.

Calculating Age using loop.

-Created an array of birthyears.
-Created an array of age which is empty.
-Created a loop.
-This loop will be used to calculate the age one by one.
-Inside the loop the push method adds data to array age will be the data of our age.

```
let birthyear =[2002,2003,2004,2006,2007,2008];
Let age = [];

for (let i = 0; i<birthyear.length;i++){
                    age.push(2022-birthyear[i])
}

console.log(age)
```

Continue And Break

-Continue Jumps over one iteration in a loop.
-Break jumps out of the loop.

```
let jonas = [`Jonas`,`Schmedinarm`,2037-1931,`Teacher`,true];
for(let i = 0;i<jonas.length;i++){
if(joans[i]===`string`){continue}
console.log(joans[i])
                                }
```

Looping backwards in Array.

-We can loop from backwards in the array where our last values will be printed first in that order.
-Our initialization expression will be defined with length of array minus one.
-We then set our condition if its >=1.

```
Let array = [`Emmanuel`,`John`,`Koech`,`Pogba`,`Chris`]

for(let i = array.length-1;i>=0;i–){
                                    console.log(array[0])}
```

Nested Loops

-This is putting a loop a child loop into the main loop,Or putting loops inside loops.
-Here as we know we expect an output.
-So we create a loop and the inside it we create another loop.

```
for(let i = 1;i<4;i++){
                console.log(`Hello${i}`)
                for (let z = 1;z<5;i++){
                ‘Manu’})
                }
```

-Here the Main loop is executed first with the child loop in it.
Output:-

Hello there1
Manu1
Manu2

Hello there2
Manu1
Manu2

Hello there3
Manu1
Manu2

While Loop

-It loops through a block of code as long as a specified condition is True.
-Unlike the for loop the condition variable is initialized outside the loop.
-While Syntax
while(condition){ code block to be executed; counter}

Here our condition is Initialized.
let i = 10;
Our condition is tested here
while(i<=10){
    console.log(`Hello${i}`);
    Increment happens here
    i++
}

Summary

For Syntax

for(initialization of condition; testing condition; increment/decrement){ statement }

for(let i =0;i<10;i++){ code to be executed}

While Syntax

while(condition){ code block to be executed; counter}

for(let i =0;i<10;i++){ code to be executed}

Math Object.

-It is used to perform mathematical tasks on numbers.

Math Properties.

-The syntax is Math.property

Math.PI -Returns PI
Math.SQRT2 - Return square root of 2.
Math.SQRT 1/2 - Return the square root of 1/2
Math.LN2 -Returns the natural logarithm of 2
Math.LN10 -Returns the natural logarithm of 10
Math.LOG2E -Returns base 2 logarithm of E
Math.LOG10E -Returns base 10 logarithm of E

Math Methods.

-The syntax is Math.method(number)

Math.random() -Returns a random number between 0 and 1
Math.trunc(x) -Returns the integer part of x
Math.round(x) -Returns x rounded off to the nearest integer.

Math.floor(x) -Returns x rounded down to its nearest integer.
Math.ceil(x) -Returns x rounded up to its nearest integer.

Scoping

-This is determining where variable,functions and objects are accessible during runtime.

Type of Scopes in Javascript.

1.Global Scope
2.Local Scope

Global Scope.

-Variable declared outside the function are globalscopes,The parent function is usually the global Scope.

Local Scope

-These are those declared inside of a block

```
This is a Global Scope
let yearofBirth = 2002;

This is a Global Scope Function
let age = ()=>{
    return 2022-yearOfBirth;

     Local Scope Function
     let countryOFBirth = (country)=>{
       return `You were born in ${country} `
}
console.log(countryOfBirth(`Kenya`))
}
```

Hoisting

-This is the behavior of using a function or variable before it is Declared.

Using a variable

-Here we have used our variable before we declared it.
-Output here will be undefined
-Error.

```
console.log(firstName);
console.log(secondName);
console.log(fullName);
var firstName = `Emmanuel Koech`;
let secondName = `Emmanuel`;
const fullName =`Emmanuel Koech`;
```

Using a function expression/declarations/Arrow functions

Function Declaration

-When this is run it will be executed

```
yourName(`Emmanuel`)
function yourName(name){
  console.log(`Hello ${name}`)
}
```

Function Expression

-Upon running this won't execute.
-Hoisting is not supported in Function expressions.

```
yourName(`Emmanuel`)
let yourName= Function(name){
  console.log(`Hello ${name}`)
}
```

Arrow Function

-Upon running this block won't execute.
-Hoisting is not supported in Arrow Functions.

```
yourName(`Emmanuel`)
let yourName= name=>console.log(`Hello ${name}`)
```

Primitive Value and Object Reference.

-How primitive values and Object Reference Affect Variable Assignments.

Primitive value

```
let lead = `Emmanuel Koech`;
let co_lead = lead;
colead = `Emmanul Kiprotich`
console.log(lead, colead);
```

-We create a variable lead.
-Then create a new variable colead and set its content to lead.
-We then reassign our value to `Emmanuel Kiprotich`.

Object Reference

```
let lead = {
        name:`Emmanuel Kiprotich`;
       group:`4PF`
}

let co_lead = lead;
colead.name = `Emmanuel Koech`;
```

-Here we assigned an object lead with different Properties.

-We then created a new object that takes the properties of the previous object `lead`.
-We then changed our property name to have a new value.

Object.assign()
-It is used for cloning an object.
-It is used to merge objects with the same properties.

```
let lead = {
        name:`Emmanuel Kiprotich`;
        group:`4PF`
}

let co_lead = object.assign({},lead);
Colead.name = `Emmanuel Koech`;
```

Destructuring Assignment

-It makes it possible to unpack values from arrays or Properties from objects into distinct variables.

Destructuring Arrays.

-Destructuring is the process of breaking an array down and extracting only what is needed.
-In destructuring the order that variables are declared is important.
-If we only wanted a car and a suv we skip truck but put a comma.

Old Way

```
let vehicles = [`Mustang`,`f-150`,`expedition`];

let car = vehicle[0];
let truck = vehicle[1];
let suv = vehicle[2];
```

New way

```
let vehicles = [`Mustang`,`f-150`,`expedition`];
let[car,truck,suv]=vehicles;
```

-Skip Truck

```
let[car,,suv]=vehicles;
```

When Arrays are in a Object

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
        categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
        starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
        mainMenu:[`Pizza`,`Pasta`,`Risoto`],
        openingHours:{
                thu:{
                 open:12,
                 close:22,
                },
                fri:{
                  open:11,
                  close:23,
                },
                sat:{
                   open:11,
                   close:23,
                }

                }
}

const [first,second,third]=restaurant.categories
```

Destructuring

-Changing Positions

```
const [first,second,third]=restaurant.categories
[first,second,third]=[third,second,first]
```

Nested Destructuring

-Here All of the Arrays will be printed.

```
let categories = [`Italian`,`Pizzeria`,[`Vegetarian`,`Organic`]];
let[first,second,third]=categories
`Italian`,`Pizzeria`,[`Vegetarian`,`Organic`]
```

-To print each Item in the array we nest our variable.

```
let categories = [`Italian`,`Pizzeria`,[`Vegetarian`,`Organic`]];
let[first,second,[third,four]]=categories
```

Destructuring Objects.

-Here we use the calibraces as we assign the variables.

-The variable is the key in our object unlike in arrays where we created our own variable and also positioned it.

-The order does not matter.

Old Way

```
let person ={
      firstName:`Emmanuel`,
      lastName:`Koech`;
}

let firstPerson =person.firstName;
let firstPerson =person.firstName;
```

New Way

```
let person ={
```

```
    firstName:`Emmanuel`,
    lastName:`Koech`;
}

let{firstName,lastName}=person
```

Changing variable Names

-Here we use a semicolon after the current variable.

-Here the current key begins then followed by a semicolon then new name.

```
let person ={
    firstName:`Emmanuel`,
    lastName:`Koech`;
}

let{firstName:nameOne, lastName:nameTwo}=person
```

Object Used in below:-

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
        categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
        starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
        mainMenu:[`Pizza`,`Pasta`,`Risoto`],
        openingHours:{
                thu:{
                 open:12,
                 close:22,
                },
                fri:{
                  open:11,
                  close:23,
                },
```

```
    sat:{
      open:11,
      close:23,
    }


}
```

Spread Operator (...)

-It allows us to quickly copy the part of an existing array or object into another array or object.
-It is usually on the right of the assignment operator.

```
let numberOne =[1,2,3,4]
let numberTwo =[1,2,3,4]
let combinedNumbers =[...numberOne,...numberTwo]
```

Rest Operator

-It is used in array destructuring to accept an indefinite number of values.
-It is also used in objects.
-It is usually on the left side of the assignment.
-It is usually at the end of the destructuring assignment.
-It is only one, not more than two.

```
Let [first,second,...rest]=[...restaurant.mainMenu, ...restaurant.starterMenu]
let{first,...rest}= restaurant.openingHours
```

-In the above we used the spread operator on the right hand side.
-In the left we used the rest operator to accept the rest of indefinite values.

Rest Operator in Functions

-It allows the function to accept an indefinite number of arguments as arrays

```
let numbers = (...numbers)=>console.log(numbers)
```

numbers(3,3,4,4,4)

Wrong

let numbers =(three,three,four,four,four)=>console.log(three,three,four,four,four);
numbers(3,3,4,4,4)

Short Circuiting

-There are three logical operators in javascript.
-These are AND,OR and NOT. &&,||,!
-Short circuiting is the evaluation of an expression from left to right with || and && operators.

Truthy Values

`0` -A string containing a single zero.
`False` -A string containing the text “false”.
[ ] -An empty array.
{} -An empty object.
function(){} -An empty function.
Numbers above 0

Falsy Values

`` -Empty String
Null
Undefined
NaN
0 -Zero
-0 -Minus Zero
False

Or Operator (||)

-It circuits when the value is a truthy value.

```
console.log(null || undefined || 20 || 30 || `0`)
```

-Here all truthy values are supposed to execute but it's only the first that will execute.
-The first truthy value in the circuit will be executed.

And Operator

-It circuits when the value is a falsy value.

```
console.log(null || undefined || 20 || 30 || `0`)
```

-Here all falsy values are supposed to execute but the first falsey value will execute.
-The first truthy value will execute 20.

Nullish coalescing operator ??

-It is written as two question marks ??
-It returns the first argument if it's not null/undefined.

Logical Operator

```
let rest1 = {
        Name:”Capri”,
        numGuest:20,
}

let rest2 = {
        Name:”la Piazza”,
        owner:`Emmanuel Koech`
}
```

-We can use the or Operator to add numGuest Property to rest 2, Since when it is circuit it will return a truthy value. Numbers are truthy Values.

```
rest2.numGuest = rest2.numGuest || 10;
rest2.numGuest  ||= 10
```

For Loop.

-Normally we used the for loop to iterate through an array.
-In ES6 a new and easier way was introduced.

Previously.

-Return with the index number

```
let arr = [10,10,10,11,13]
for(let i = 0; i<arr.length;i++){
    console.log(arr[i])
}
```

-When using a for loop.
-ES6 way.
-To return the index number we use a method called .entries()

```
let arr = [10,10,10,11,13]
for(const items of arr)console.log(items)

let arr = [10,10,10,11,13]
for(const items of arr.entries())console.log(items)
```

Es6 Enhanced Object literals

-In Es6 a new way to assign objects into existing objects was introduced.
-The first way is the old way,Where we call our existing object with the key and also assign it the value name.
-In Es6 we call the object with the key then comma only.

-Here are 2 objects Restaurant and Opening Hours

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
        categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
        starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
```

```
        mainMenu:[`Pizza`,`Pasta`,`Risoto`]
}

const openingHours={
                        thu:{
                         open:12,
                         close:22,
                        },
                        fri:{
                          open:11,
                          close:23,
                        },
                        sat:{
                            open:11,
                            close:23,
                        }
```

Creating One Object the Old Way

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
        categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
        starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
        mainMenu:[`Pizza`,`Pasta`,`Risoto`],
        openingHours:openingHours
}
```

Creating One Object the New Way

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
```

```
    categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
    starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
    mainMenu:[`Pizza`,`Pasta`,`Risoto`],
    openingHours,
}
```

Arrays,Objects,functions

-In Es6 these can be written differently in object keys when they are property.
-Instead of hard coding the weekdays again in the object we can call them with the index numbers from the arrays.
-Also the we declare function we need not to use the function key word,Also no need to separate the key with colon

```
const weekdays =
[`Monday`,`Tuesday`,`Wednesday`,Thursday`,`Friday`,`Saturday`,`Sunday`];

const openingHours={
                    [weekdays[3]:{
                        open:12,
                        close:22,
                    },
                    [weekdays[4]]:{
                        open:11,
                        close:23,
                    },
                    [weekdays[5]]:{
                        open:11,
                        Close:23,
                    },
                    time(timeNow){
                        return ‘Hello World’;
                    }
}
```

Optional Chaining ?.

-The optional Chaining operator accesses an object property,If the object property is unavailable it returns null or undefined.
-Optional Chaining operator can be used with nullish operator ??
-Instead of returning undefined it will return the value we assign after assigning the nullish operator.
-When we call the function the function name comes first then the optional chaining operator then the brackets yourfunction?.()

```
const adventurer = {
                name:`Emmanuel`,
                cat:{
                        name:`Dinah`
                                }
                yourName:function(name){
                                return “I am ${name”
                                }

};
```

let dogObj = adventurer.dog?.name??”No such input”;
-No such input prints here because there is no property such as a dog.

let yourNameObj = adventurer.yourName?.(“Emmanuel”);
-When logged “I am Emmanuel prints” since the property name is available in the object.

Looping Over Objects

-We can loop through the object property names(Key).
-We can loop through the object property values.
-Finally we can loop over an the whole object

Object.keys()

-It is used to access the property name of the object (key).
-Inside the bracket it is where we access the name of our object.
-We use it to loop the object key.

Object.value()

-It is used to access the property value of the object(Object values).
-Inside the bracket it is where we access the name of our object.

Object.entries()

-It is used to access both the key and value of an object.
-It returns an array whose elements are string paired properties [key,value] .
-Hence we can destructure it.
-An object is made up of key and value.

```
const restaurant ={
        name:`Classico Italiano`,
        location:`Via Angelo Tavaanti 23,Firenze,Italy`,
        categories:[`Italian`,`Pizzeria`,`Vegetarian`,`Organic`],
        starterMenu:[`Focaccia`,`Bruschetta`,`Garlic Bread`,`Caprese Salad`],
        mainMenu:[`Pizza`,`Pasta`,`Risoto`],
        openingHours:{
                thu:{
                 open:12,
                 close:22,
                },
                fri:{
                  open:11,
                  close:23,
                },
                sat:{
                   open:11,
                   close:23,
```

}

}

-Here we have accessed the object.

let openingHourskey = object.key(restaurants.openingHours)

let openingHoursValue = object.values(restaurants.openingHours)

let openingHoursEntries = object.entries(restaurants.openingHours)

-Here we loop through the object.

for (const key of openingHoursKey)console.log(key)

for (const value of openingHoursValue)console.log(value)

for (const [key,value] of openingHoursEntries)console.log(object)

Sets

-Sets are collections of unique values,They are almost like arrays.

-We define a set using new set().

-It can hold any data type.

-Each value can occur only once in a set.

-Sets are iterables

-Main use case of set is to remove duplicate values from an array.

Set Methods

add() -Add new value to the set, New value goes inside the bracket.

delete()-Removes value, Value we want to delete goes inside the bracket.

has()-Checks if the value exists in the set,Value we want to check goes in the bracket.

clear()-It removes all the values in the set.

size - Checks the size of set

let fruits = new

Sets([`Pineapple`,`Apple`,`Oranges`,`Avocado`,`Pears`,Pineapple`,`Apple`,`Oranges`])

fruits.add("Watermelon")

fruits.delete(“Oranges”)
fruits.has(“Orange”) If it is there it will return true else false if missing.
fruits.clear()
size

Removing Duplicate Values from arrays

Let arr =
[`Pineapple`,`Apple`,`Oranges`,`Avocado`,`Pears`,Pineapple`,`Apple`,`Oranges`]

Let arrNonDup = new set([...arr])

Map

-This is a collection of keyed data items,just like an object,Difference is that it allows keys of any Data type.
-We declare our variable then assign it to new Map().
-Afterwards we we can store our data in the variable we declared variable.set()
- .key() and .values() access the map keys and values respectively.
-In the set method the first index is the key then the second one is the value
set(key,value).

Methods

variable.set() -Stores the value by key.
variable.get() -Returns the value by key.
variable.delete() -Removes the values by key.
variable.clear()-Removes everything in the map.
variable.has()-Used to check if the key exist, If it does true is returned else false returned
variable.size -Returns the size of the map.

let map = new Map();
map.set(“String”,”Hello”,”World”);
map.set(1,”Number”);

```
map.set(true,”Boolean”);

map.delete(string);
map.has(“arr”);

console.log(map.get(set))
```

Better way to assign Map

-We assign our variable with new Map().
-Afterwards we put the value with its key in an Array.
-We do not use sets
-We use arrays..
-Maps are iterables

```
let map = new Map([
        [“String”,”Hello”,”World”],
        [1,”Number”],
        [true,”Boolean”],
);
```

Looping through Maps

-Maps are iterable.
-Here is a way to loop through Maps.

```
for (let [key,value] of map){
      console.log(key,value)
}
```

Strings

-These are characters inside the quotes. “String”

String Methods

length

-It Checks the length of the string.

```
let name = “Emmanuel”
name.length
```

indexOf()

-It returns the first occurrence of a value in the string.

```
let name = “Emmanuel”;
name.indexOf(“m”)
```

1 is the output since the first m is the first occurrence .

lastIndexOf()

-It returns the last occurrence of the value in the string.

```
let name = “Emmanuel”;
name.indexOf(“m”)
```

2 is the output since it is the last occurrence .

replace()

-Replaces characters in string. First character is in string,Second is new.

```
let name = “Emmanuel”;
name.replace(“Emmanuel”,”Koech”)
```

includes()

-It checks if a string contains certain characters.

-When using conditions you should use toLowerCase to get accurate condition.

-If the characters are there it returns true if not it returns false.

```
let name = “Emmanuel”;
name.includes(“Emmanuel”)
```

True will be the output.

startsWith()

-It checks if a string starts with certain characters.

-If the characters are there it will return true else false.

```
let name = “Emmanuel”;
name.startWith(“Emm”)
```

True will be the output.

endsWith()

-It checks if a string ends with certain characters.

-If the characters are there it will return true else false.

```
let name = “Emmanuel”;
name.endsWith(“el”)
```

True will be the output.

toUpperCase()

-Returns the entire String to Uppercase when Converted.

```
let name = “Emmanuel”;
name.toUpperCase()
```

toLowerCase()

-Returns the entire String to Lowercase when Converted.

```
let name = “Emmanuel”;
name.lowerCase()
```

trim()

-Returns the string with removed whitespaces.

```
let name = “  Em  m  anuel”;
name.trim()
```

Emmanuel will be the output

slice()

-It Extracts part of a string and returns a new string. -1 is last part of string

-It can take the index number of a string from where extraction begins and Ends.

-Other methods can be used inside it. indexOf() , lastIndexOf().

```
let name = “Emmanuel”;
name.slice(‘0’,’2’)
name.slice(name.indexOf(`m`),name.lastIndexOf(`u`))
```

split()

-It splits a string into an array.

-It will split the string into an array where there is a character we set.Eg ,+-

```
let meat = “pork,beef,mutton,chicken”
meat.split(,)
```

join()

-It is used to join the elements of an array into string.

-The elements of the string will be separated by default value by comma.

-Other separators can be used in the bracket()

```
let meat = [pork,beef,mutton,chicken];
meat.join(`any separator eg -,+`)
```

repeat()

-It returns a string multiple times.

```
let name = “Emmanuel”;
name.repeat(4)
```

-Emmanuel will repeat 4 times.

Array Methods

Slice()

-It extracts a piece of an array into a new array.

-Slice is immutant,It does not mutate original original array

let letters = [`a`,`b`,`c`,`d`,`e`];

letters.slice(2) - [`c`,`d`,`e`]
letter.slice(2,3) - [`c`,`d`]
letter.slice() - [`a`,`b`,`c`,`d`,`e`] -When no index set it returns duplicate.
console.log(letters) -All original arrays will be returned.

Splice()

-It extracts a piece of an array into a new array.
-Slice is mutant, It mutates the original array.

let letters = [`a`,`b`,`c`,`d`,`e`];
letters.slice(2) - [`c`,`d`,`e`];
console.log(letters) - [`a`,`b`] -The arrays which were not extracted will be returned.

reverse()

-It reverses the order of elements in arrays.
-Reverse mutates the original array.

let letters = [`a`,`b`,`c`,`d`,`e`];
letters.reverse() -[`e`,`d`,`c`,`b`,`a`]
console.log(letters) - [`e`,`d`,`c`,`b`,`a`] -The original array order is overwritten.

concat()

-It joins two or more arrays together, Just like the spread operator.
-The method does not alter the original array.

let num1 = [`1`,`2`,`3`,`4`,`5`];
let num2 = [`6`,`7`,`8`,`9`,`10`];
let num3 = [`11`,`12`,`13`,`14`,`15`]

let allNum =num1.concat(num1,num2,num3);

Results:- [`1`,`2`,`3`,`4`,`5`,`6`,`7`,`8`,`9`,`10`,`11`,`12`,`13`,`14`,`15`]

Easier way is using spread operator
Let allNum = [...num1,...num2,...num3]
Results:- [`1`,`2`,`3`,`4`,`5`,`6`,`7`,`8`,`9`,`10`,`11`,`12`,`13`,`14`,`15`]

join()
-It returns an array into a string.
-A separator can be specified , By default its (,)

let num1 = [`1`,`2`,`3`,`4`,`5`];
num.join(``) - 123456
num.join(`+`)- 1+2+3+4+5

length()
-Checks the length of Array
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
Items.length - 4

push()
-Adds element at end of array

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.push(“buffalo”)- [“cows”,”dog”,”goat”,”pigs”,”hen”,”buffalo”]
Buffalo will be added in the array

Pop()
-Removes the last element of an array and also prints it in the terminal.
let items =[“cows”,”dog”,”goat”,”pigs”,”hen”];
items.pop() -[“cows”,”dog”,”goat”,”pigs”];

Hen will be removed from the array.

shift()

-Remove the first element of an array.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.shift()- [”dog”,”goat”,”pigs”,”hen”]
cows will be removed from the array

unshift()

-Adds first element of an array but overwritten it

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
items.unshift(“buffalo”) -[“buffalo”“cows”,”dog”,”goat”,”pigs”,”hen”]
buffalo will be added from the array

splice()

-It is used to remove data in an array or add data from a certain index number.
let items = [“cows”,”dog”,”goat”,”pigs”,”hen”]
items.splice(2) -[“cows”,”dog”]
-Here all items from index 2 will be removed.

items.splice(2,3) - [`cows`,`dog`,`hen`]
-Here all items from index 2 to 3 will be removed the rest remains

items.splice(2,3 11, 12) - [[`cows`,`dog`,`hen`,11,12]
-Here all items from index 2 to 3 will be removed the rest remains but 11,12 will be added in the array as we said splice is used to add data in arrays also.

indexOf()

-It returns the indexOf the value specified from the array.
-If the value specified is not found it returns -1
-It searches from left to right.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`cows`))

0 will be the return value at the terminal.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`buffalo`))
-1 will be the return value at the terminal since there is no such value in our array.

Includes()
-Returns either True or False if the value is in the array.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.include(`cows`))
True will be the return value at the terminal since cow is present in the array.

let items =[“cows”,”dog”,”goat”,”pigs”,”hen”]
console.log(items.indexOf(`buffalo`))
False will be the return value at the terminal since there is no such value in our array

Default Function Parameter.

-In Javascript, Function parameters generally have a default value of undefined.
-This means if you don't pass any arguments when called parameters will have default values

```
const createBooking = (flightNum,numPassengers,price){
        Const Booking={
                        flightNum,
                        numPassengers,
```

```
                price,

}
}
createBooking()
```

-Here the value printed will be undefined.

```
const createBooking =
        Es6 Way
(flightNum='003',numPassengers=20,price=numPassenger*200){

        Es5 way
         flightNum=flightNum || 20;
         numPassengers = numPassenger || 30;
         price=price || 40;


         Const Booking={
                        flightNum,
                        numPassengers,
                        price,

}
}
createBooking()
```

-In above we have set values for our default parameters.
-Es6 way is the modern way while Es5 Way is the old Way.

First Class and Higher order Functions

-Functions is Javascript are first class citizen,This means you can:-
1.Assign them to variables.

```
const hiFunction = ()=>`Helloworld`;
hiFunction()
```

2.Pass a function as an argument to another function.

```
const hiFunction = ()=>`Hello World`;
hifunction(function(){return “Hello World”})
```

3.Return a Function from another Function.

```
const hiFunction = (greeting)=>{
        return function(name){ ` ${greetings},${name}`
}
hiFunction(`Hello`)(`Manu`)
```

-Higher Order functions are functions that use other functions arguments or return functions.

Call And Apply Method

-Call is a Javascript method that is used to call methods of various objects containing this keyword.
-With call You can write a method that can be used in different objects.
-The key names should be the same.
-Remember to store the object method in a separate variable outside the object reference.
-Apply is same as Call difference is function arguments go into the [], Stored in arrays.
-Call syntax

```
call(objectName,  function arguments goes here)
```

-Apply syntax

```
apply(objectName, [ function arguments goes here])
```

First Object with method

```
const flight = {
        airline:`Kenya Airways`,
        firstName:`Emmanuel`,
        secondName:`Koech`,
        bookinfo:[],
        bookings:function(){
                console.log(`Hello ${this.firstName} ${this.secondName} be ready to fly
with $(this.airline)`
        this.book.push(this.firstName, this.secondName,this airline)
)

}

}
```

Booking Method Stored in Variable.

```
const bookings = flight.booking
```

Second Object

```
const flight2 ={
        airline:`Kenya Airways`,
        firstName:`Emmanuel`,
        secondName:`Koech`,
        bookinfo:[],

}
```

Using Call method

```
bookings.call(flight2)
bookings.apply(flight2, [])
```

Bind Method

-It Allows an object to borrow another method from another object.
-The main object as the method.
-The second object borrows the method object from the main object.
-Bind takes the second object as its arguments.

First Object with method

```
const flight = {
        airline:`Kenya Airways`,
        firstName:`Emmanuel`,
        secondName:`Koech`,
        bookinfo:[],
        bookings:function(){
                console.log(`Hello ${this.firstName} ${this.secondName} be ready to fly
with $(this.airline)`
        this.book.push(this.firstName, this.secondName,this airline)
)

}

}
```

Second Object

```
const flight2 ={
        airline:`Kenya Airways`,
        firstName:`Emmanuel`,
        secondName:`Koech`,
        bookinfo:[],

}

let flight2Bind = flight.bookings.bind(flight2);
flight2Bind()
```

Closures

-It means the inner function has access to the variable and parameters of its outer function even after the outer function is returned.

```
function OuterFunction(){
        var outerVariable = 1;
        function InnerFunction() {
                  alert(outerVariable);
                }
                InnerFunction();
}
```

-Here the inner function can access the outer function variable even when executed separately

```
const bookings = ()=>{
                let passengerCount = 0
                return  function(){
                        passengerCount++
                        console.log(`${passengerCount} passenger`)
}
}

const  booker = bookings();
booker()
```

SetTimeout

-Its a javascript method that executes a function after the timer expires.

Syntax

setTimeout(Functions goes here, Then the time)

At Method Arrays

-It is an es6 syntax that returns an element of an array by its index number.
-It also works in strings.

Old way

```
Let arr = [`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];
console.log(arr[2])
```

New Way

```
Let arr = [`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];
console.log(arr.at(0))
```

forEach Loop

-For each is an iterative method used in an array,It calls a provided callback function.
-The forEach has a function call back,which is an anonymous function .
-The array comes first followed by for each method.
-Inside the method is the function, Arrow functions can be used.
-The parameters of the function carry the (current element, index , entire array).
-Above is the correct order.

Syntax of forEach loop

```
arrayName.forEach(anonymous function)
```

Example

```
let cars = [`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];

cars.forEach((car ,index,array)=>{
  let number = car + 1
  console.log(`This is my ${number} like my ${car}`)
})
```

NB:-You cannot break out in the ForEach Loop, No Continue and Break Statement.

forEach in Maps

-It can iterate a map .
-We map comes first followed by the forEach method.
-Inside our Method is the anonymous function.
-The function parameters carry (value,key,map).

Syntax.

mapName.forEach(function)

Example

```
let currencies = new Map([
   [`USD`, `United States Dollar`],
   [`EUR`,`Euro`],
   [`GBP`,`Pound sterling`]
])

currencies.forEach((value,key,map)=>{
     console.log(`This is the value: ${value}
                        This is the Key:${key}`)
})
```

forEach in Sets.

-Here we call the set name followed forEach method.
-Inside our method we call the function.
-Set does not have an index hence inside our function we start with the value.
-Where we are supposed to put the key we put a _ .

Syntax

setName(forEach(function))

Example

```
let currenciesUnique = new Set([`USD`,`GBP`,`USD`,`EUR`,`EUR`]);
currenciesUnique.forEach((value,_,set)=>{
    `This is my value ${value}`})
```

Map Method

-It iterates over an array and returns a new array.

-Its main difference from for loop is that it creates a new array.

-In forEach for us to create new array we have to use the push method to an empty created array variable.

Syntax

```
arrayName.map(function(value,index,array){})
```

Example Using Map

```
const cars =  [`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];
cars.map((car,index)=>{
        console.log(car)
})
```

New array will be returned.

```
[`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];
```

Example Using forEach

```
const cars =  [`Benz`,`Audi`,`Toyota`,`Bentley`,`Volgswagen`];
const newCarArray = []
cars.map((car,index)=>{
        newCarArray.push(car)
})
```

-Here we had to use the Push method to push the car to the new car array method.

Filter Method

-It is an iterative method in the array that creates a new array of elements that pass a test provided by a function.
-It does not change the original array.

Syntax

```
arrayName.filter(function(value,index,arr),thisValue )
```

Example

```
const movement = [200,450,-400, 3000,-650,70,1300];
const deposit = movement.filter((mov,index)=>{
        return mov>0;
        })
console.log(deposit)
```

OutPut:- [200, 450, 3000, 70, 1300]

Reduce Method

-It is used when you have array values and you want to add them all up.
-It loops through the array hence it is an iterator.
-It has 4 parameters (accumulator,value,index,array).
-The accumulator is the sum of all array elements.
-The value is each iterator in the array.

Syntax.

```
arrayName.reduce(function(accumulator,value,index,array){0})
```

Example.

```
const movements = [200,450,-400, 3000,-650,70,1300];
const balance = movement.reduce(total,value,index){
        return total+value,0
}
console.log(balance)
```

Chaining Methods

-Method chaining is calling a method on another method without assigning it multiple times in a variable.
-So far we have learnt three data transformation methods : map,filter and reduce.
-Map loops through an array and returns a new array.
-Filter will loop through an array and remove the values we set to filter.
-Reduce will return the sum of all values in arrays.

Syntax

```
const arr = [values];
const newArr = arr.method.method.method
```

Example

```
const movement = [200,450,-400, 3000,-650,70,1300];
let movementArr = movement
                        .filter(move=>move>0)
                        .map(move=>move*1.3)
                        .reduce((accu,value)=>accu+   value)
```

Find Method.

-It is an iterative method that returns the first value of an Array when a certain condition is satisfied.
-It works the same as the filter method but returns the first value of the array.
-The difference between find and filter is that filter returns a new array of all the conditions satisfied from the array whereas find returns only the first value from the array which the condition satisfies.No array is returned in find.

Syntax.

```
const arrs = [values];
const newArr = arrs.find(function(arr,index,array));
```

Example.

```
const movements = [200,450,-400, 3000,-650,70,1300];
movements.find(movement=>movement<-1)
```
Output:-400

Example

```
const account1 = {
        owner:`Jonas Schmrdtmann`,
        movement:[200, 450. -400, 3000 , -650, -130, 70 ,1300],
        interestRate: 1.2;
        pin:1111
}
```

```
const account2 = {
        owner:`Jessica Davis`,
        movement:[5000, 3400. -150, -790 , -650, -3210, 1000 ,8500, 30],
        interestRate: 1.5;
        pin:222
}

const accounts = [account1,account2];
accounts.find((acc=>acc.owner === `Jessica Davis` ));
```

Output = The second Array.

```
        {
        owner:`Jessica Davis`,
        movement:[5000, 3400. -150, -790 , -650, -3210, 1000 ,8500, 30],
        interestRate: 1.5;
        pin:222
        }
```

-Here the FindIndex loops through the accounts array.
-AfterWards we check for the account owner if it satisfies the conditions which is set.
- 1 is the output since the owner is in the second array which is account 2.

FindIndex

-It returns the index of the first element of an array that satisfies the testing function.
-If no element satisfies the testing condition -1 is returned.

Syntax

```
const arrs =[]
arrs.findIndex(function(arr,index,array))
```

Example

```
let arrs = [200,450,-400, 3000,-650,70,1300];
arrs.findIndex(arr=>arr);
Output 0
```

Example

```
const account1 = {
        owner:`Jonas Schmrdtmann`,
        movement:[200, 450. -400, 3000 , -650, -130, 70 ,1300],
        interestRate: 1.2;
        pin:1111
}
const account2 = {
        owner:`Jessica Davis`,
        movement:[5000, 3400. -150, -790 , -650, -3210, 1000 ,8500, 30],
        interestRate: 1.5;
        pin:222
}
```

const accounts = [account1,account2];

accounts.findindex((acc=>acc.owner === `Jessica Davis`));
Output = 1

-Here the FindIndex loops through the accounts array.
-AfterWards we check for the account owner.
-1 is the output since the owner is in the second array which is account 2.

Some method

-It is an iterative method.
It checks if at least one array element passes the given test condition.-
-It returns true if atleast one element passes the test.
-It has a call back like the rest of the methods we have seen.
-It returns False if all the elements fail the test.

Syntax

let arrs = [];
arrs.some(function(element,index,array))

Example

let arrs = [200,450,-400, 3000,-650,70,1300];
arrs.some(arr=>arr>5)
OutPut:- True

Every Method

-It is an iterative method.
-It checks if all arrays in an element passes the given test condition.
-It returns true if all the elements in the array satisfies the test condition.
-If Not all satisfy the condition false is returned.

Syntax

let arrs = [];

arrs.some(function(element,index,array))

Example

let arrs = [200,450,-400, 3000,-650,70,1300];

arrs.every(arr=>arr>5)

OutPut:- False

-False since not all are greater than 5.

Flat Method

-It is an ES2019 array method that is used to reduce nesting of an array/unnest arrays.

-It does not accept a call back function.

-It only accepts one parameter when we want the unnest deep array.

-The parameter Specifies how deep a nested array structure should be unnested.

Syntax.

array.flat(depth)

Example.

let num = [1,2,3,4,5,6,7,[8,9]];

console.log(num.flat())

Output:[1,2,3,4,5,6,7,8,9]

let num2 = [1,2,3,4,5,6,7,[8,9,[10,11,12,[13,14]]]];

console.log(num.flat(3))

Output:[1,2,3,4,5,6,7,8,9,10,11,12,13,14]

FlatMap Method

-It returns a new array followed by flattening the array to one level,Unesting the array.

-Unlike Flat method this method does not deep nest an array.

-It is a combination of flat and map methods.

-A recap the map method iterates through the array and returns a new array.

Syntax

flatMap(function(element,index,array))

Example

let array= [1,2,3,4,5,6,7,[8,9]]

array.flatMap(arr=>arr)

Output: [1,2,3,4,5,6,7,8,9]

Sort Method

-It is used to sort the array in place in a given order.

-It sorts the array of string elements, integer elements.

-By default it follows the ascending order.

-Sort can be used with a function call back.

Syntax

Main Syntax

arrayName.sort(function())

Recommended Syntax

arrayName.sort((a,b)=>a-b) - Ascending order

arrayName.sort((a,b)=>b-a) - Descending order

Other Syntax but not recommended

arrayName.sort() -By Default it is in ascending order.(small to large)

arrayName.sort().reverse() -This sorts arrays in descending order.(large to small)

Example 1

Without Call Back Function

-This will return the array in ascending order.

let array = [20,5.2,-120,100,30,0]

array.sort()

Output:-[-120,0,100,20,30,5.2]

We Should use the callback function since this brings problems

-This will return the array in descending order.

let array = [20,5.2,-120,100,30,0]

array.sort().reverse()

Output:-[5.2,30,20,100,0,-120]

With Call Back Function

-This will return the array in ascending order.

let array = [20,5.2,-120,100,30,0]

array.sort((a,b)=>a-b)

Output:-[-120,0,5.2,20,30,100]

-This will return the array in descending order.

let array = [20,5.2,-120,100,30,0]

array.sort((a,b)=>b-a)

Output:-[-120,0,5.2,20,30,100]

Fill Method

-The fill method fills a specified element in an array with a value.

-It mutates the original array.

-The start and end position must be specified.

Syntax.

arrayname.fill(value, start,end)

Example.

let arr = [1,2,3,4,5,6];

arr.fill("three",1,3)

Numbers

-These are values that represent floating numbers like 37 or -9.25
-Values of other types can be converted to numbers using Number() function.
-Base Ten are numbers between 0-9
-Binary base 2 is number 0 1.

Number Conversion

-Strings can be converted to Numbers.
-Number() and + can be used during this conversion.

Syntax

Number()
+"String Number to be converted"

Example

let convertNumber = Number(`23`);
let converNumber = +23;
Output:- 23 23

Number parsing

-It parses a value as a string and returns the first integer.
-There are four parsing functions parseInt,parseFloat,isNaN,isFinite.
-parseInt gets rid of symbols that are not numbers in the string.
-parseFloat gets rid of symbols that are not numbers in the string.
-isNan checks if a value is Not a Number,It converts value to number before testing it.
-isFinite checks if a value is a number, It returns true if the value is a number
-isInteger checks if a value is a integer, It returns true if the value is a number

Note:-

-In parseInt, parseFloat the first value must be a number else if it's a string NaN is returned.
-Writing its syntax without the Number function is the traditional way.

Syntax

Number.parseInt(string, radix)
Number.parseFloat(string, radix)
Number.IsNaN(value)
Number.IsFinite(value)

Example 1

```
let convertNumber = parseInt(`12Hello`);
let convertNumber = parseInt(`1Hello2`);
let convertNumber = parseInt(`Hello12`);
let convertRadius = parseFloat(`1.234rem`);
```

Output:- 12 1 NaN 1.234

Example 2

```
let checkNumber = Number.isNaN(23);
let checkNumber = Number.isFinite(23);
let checkNumber = Number.isFinite(23.34);
```

Output:- false true false

Math Javascript

-It allows you to perform mathematical tasks on numbers

Math Method

Math.sqrt -It returns the square root of a number.
Math.max -It returns the highest value,It also has type coercion.
Math.min -It returns the lowest value,It also has type coercion.
Math.PI -It is a constant method that finds the value of pie in a circle.
Math.trunc - It removes any decimal part from a number
Math.round - It rounds the number to the nearest integer

to.fixed -it rounds the string to a specified number of decimals.

Example

Remainder Operator

-It returns the remainder left over when a value is divided by another value.

Syntax

%

Example

let = 5&3

Output:-2

Numeric Separator.

-It helps improve readability for all kinds of numeric literals, Underscore is used.

-The separator cannot be used at the beginning of the numeric literal.

-Also the separator cannot be used at the end of numeric literals

-The separator cannot be used with decimals.

-When the numeric separator is used in a string number during conversion it wont work.

Syntax

_

Example

let number = 1000000202;

let number = 1000_000_202;

Output:-1000000202

Output:-1000000202

BigInt

-It is used to store big integer values that are too big to be represented by a normal Javascript Number.

-It stands for big integer and was introduced Es2020.

-bigInt can be created using the bigInt function or n which will come at the end of the number

-Bigint numbers work together during mathematical operations.

-Regular numbers will not work together with bigInt during mathematical operations.

Syntax

bigInt()

n

Example

let number = 29999999999999999904;

let number = bigInt(29999999999999999904);

let number = 2999999999999999999904n;

Output:-29999999999999999668352n

Output:-2999999999999999999904n

Output:-3e+24;

Javascript Dates

-The javascript date object can be used to get year, month and day.

-By default it return the universal date and time of a certain region

-The date method can take a date string,It will automatically set it in the correct format.

-If we put a string that is not a string an invalid output will be our results.

-Dates can be converted to numbers with number objects.

Date constructors

Date()

Date(datestring)

Date(year,month,day ,minutes,seconds,milliseconds)

Example

new Date()

new Date(`December 2020 12`)

new Date(`12 December 2020:50:40:404`)

Output:-Current Location Time

Output :- 12-12-2020:00:00.000

Output :- 12-12-2020:50:40.404

Syntax

new Date()

Date Methods

-These are the common date methods.Normally it is great to store the new Date() constructor in a variable then call the methods.

Syntax

getFullyear()
getMonth()
getDay()
getHours()
getMinutes()
getSeconds()

getFullyear()

-It returns an integer value that represents the year on the basis of local time.

getMonth()

-It returns an integer value between 0-11 that represents the month on the basis of local time.

getDay()

-It returns the integer value between 0-6 that represents the day of week on basis of local time

getHours()

-It returns the integer value between 0-23 that represents the hours on the basis of local time.

getSeconds()

-It returns the integer value between 0-60 that represents the seconds on the basis of local time

toISOString()

-It returns the date object as a string format in International Standards.

Example

```
let date = new Date(2020-12-1:40:40:304);
```

date.getFullyear() **Output:**-2020

date.getMonth() **Output:-**12

date.getDay() **Output:-**1

date.getHours() **Output:-**40

date.getMinutes() **Output:-**40

date.getSeconds() **Output:-**304

date.toISOString() **Output:-**2020-12-1T40:40.304Z