History of Javascript

History:- In 1995, A Netscape (browser) programmer named Brandan Eich developed a scripting language in just 10 days.

Originally name (first name):- Mocha
Second name :- LiveScript

At that time java is famous programming language. So, for marketing purpose LiveScript changed into javascript.

★ Java and Javascript both are different programming language. nothing is common.

Mocha → LiveScript → Javascript

In 1997, there is another famous browser that was internet Explorer (Microsoft browser).
Then, Microsoft copied javascript features made own language named as JScript.
In Browser war (Netscape vs internet explorer)

Netscape → Javascript
Internet Explorer → Jscript

EcmaScript is born....

Ecma International :- Ecma international is an industry association founded in 1996, dedicated to the standardization of information and communication systems.

JavaScript + Ecma (Rules) → EcmaScript.

Problem solved :- We can implement scripting language for different browser.

same

First Ecmascript.

ES1 → 1997

ES5 → 2009 (lots of new features)

ES6 (ES 2015) → 2015 (Biggest update for JS)

ES6 is also known as Modern JavaScript

Ecma have a technical community known as Tc39 had decided that after 2015. we release javascript with new featurs every year (Annual release).

@programmer-girl--

JavaScript Features.

Features:-
Case sensitive
Dynamically typed
cross-platform @programmer-girl--
Interpreted
object-oriented Scripting language
Backward Compatible

JavaScript variables

Variables:- variables stores the data which can be changed or used when we need.
There are there *keywords to declare a variable.
keywords are the predefined words in programming languages.

- var `var name =10;`
- Let `let name =10;`
- const `const pi = 3.14;`

Datatype in JavaScript

There are two types of Data

1. Primitive
2. Non-primitive.

Primitive datatypes are :-

- Number
- Null
- String
- Bool
- Undefined
- Bigint
- Symbol

@programmer-girl--

Non-Primitive datatypes are:-

- Array
- object
- RegExp.

JavaScript Hacks

1. Convert string to numbers

Put the pulse (+) before the string
For Example:-

```
let str = "9";
console.log(typeof(+str));
```

2. Convert number into string

Add a empty string with the number
For Example:-

```
let num= 10;
console.log(typeof(num+" "));
```

@programmer-girl--

JavaScript String.

String:- String are used to store textual form of data like word, sentence. It follows zero based indexing.

```
let str = "pro";
let str = 'pro';
let str = `pro`;
```

JavaScript String Method

trim()	Slice()
CharAt()	tostring()
concat()	Substring()
indexof()	toUpperCase()
lastIndexof()	tolowerCase()

@programmer-girl--

Undefined in Javascript

- Accessing an uninitialized variable returns undefined.

```
let str;
console.log(str); //undefined
```

- Accessing a non-existing property of an object returns undefined.
- Accessing a out-of-bounds array element returns undefined

Null in JavaScript

- null means 'no value' assign to variable.
- typeof null returns 'object'
- Null is treated as false value.

@programmer-girl--

JavaScript BigInt

BigInt :- BigInt is a primitive Datatype which is used for large numeric values it doesn't represent decimal values.
It is used to represent values greater than $2^{53}-1$.

Declaration of BigInt

- By appending n at the end of numeric values.

```
var num= 9876543219865252772n;
```

- By passing the values as an argument to the BigInt().

```
var num= BigInt(987654321986525277);
```

@programmer-girl--

JS Ternary Operator

Ternary operator:- It is also called conditional operator.

→ It takes three operands.

→ It makes the code more concise.

Syntax:-

```
let variableName = Condition ? True : False ;
```

If the condition is true expression after ? will executes. If it is false, expression after : (colon) will executes.

For Example:-

```
let age = 18 ;
let warning;
age >= 18 ? (warning = "You can play")
        : (warning = "You cannot play");
console.log(warning);
```

output:- You can Play.

JS Boolean Data Type

Boolean :- It can hold only two values: true and false.

For Example:-

```
Var Read = true;  ]- typeof(Read) ]
Var Eat = False;          Boolean
```

Boolean values also come as a result of Comparisons.

For Example:-   @programmer-girl--

```
var x = 1, b = 4, y = 8;
console.log (b>x) //output:- true
console.log(b>y ) //output:- false
```

== and ===

== (Double equals operator) :- Known as the Equality or abstract comparison operator.

→ It compare variables, ignores datatype.

=== (Triple equals operator) :- Known as the identity or Strict comparison operator.

→ It compare variables as well datatype.

Js Truthy and Falsy Values

Truthy values:- It is a value that is considered true when encountered in a Boolean Context.

Example:-

true, {}, [], 42, "0", "False",
new Date(), -42, 12n, 3.14, -3.14,
Infinity, -Infinity.

Falsy values:- It is a value that is considered false when encountered in a Boolean context.

Example:-

undefined, null, NaN, false, " ",
0, -0, 0n (BigInt)

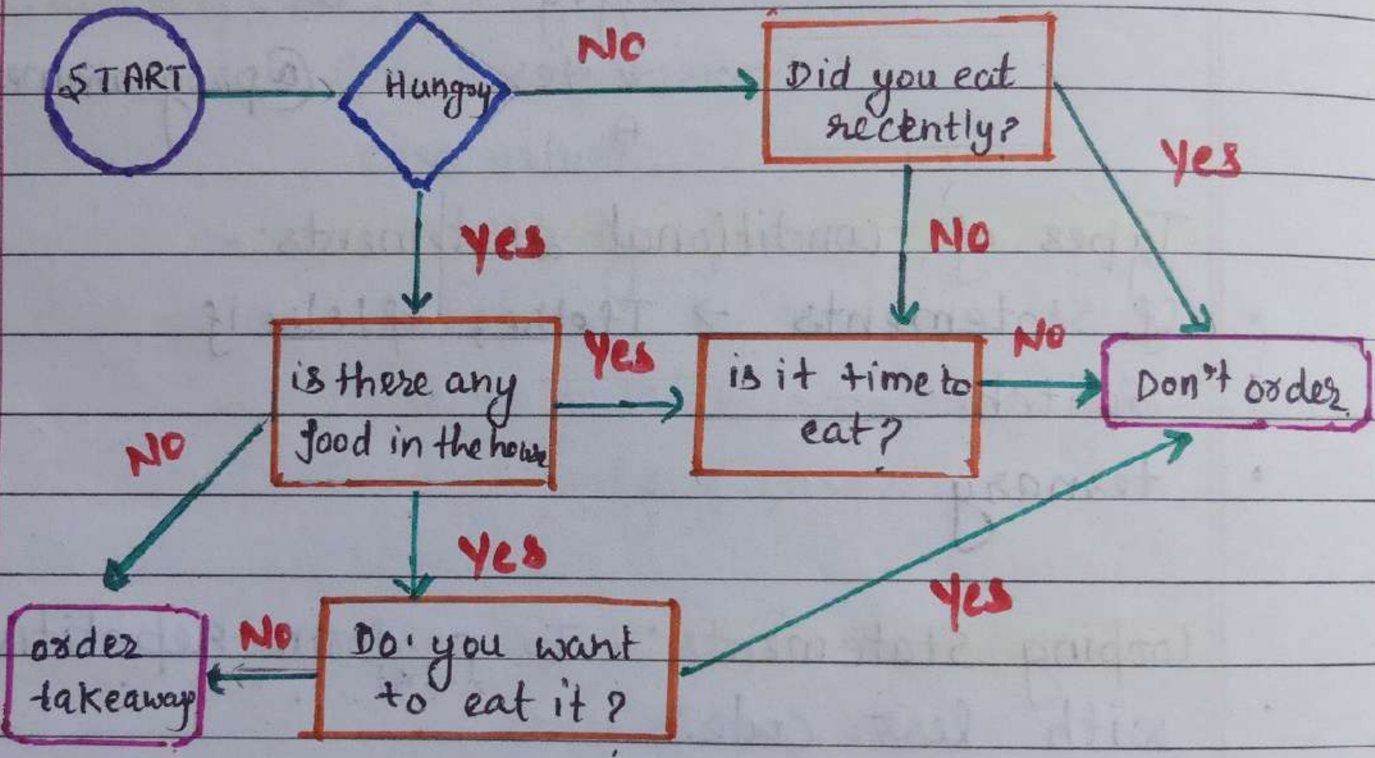
```
var values = 42;
if (values) {
    console.log(true);
}
else {
    console.log(false);
}
```

output:- true

Control Flow

Control Flow:- It allows our program to make decisions about what code is executed and when.

For Example:-



@programmer-girl.

Control Flow have two types of Statements
1. Conditional Statements.
2. looping statements.

Conditional statements:- Conditional statements are basically checks to see if a certain condition is either true or false. If the condition is true then run code A, if it's false then rund code B.

$$\begin{tikzcd} \text{Hungry} \arrow[r, "\text{NO}"] \arrow[d, "\text{Yes}"] & \text{B} \\ \text{A} & \end{tikzcd}$$

@programmer-girl--

Types of conditional statements:-

- If statements → Ifelse, if/else if
- Switch
- ternary

Looping Statements:- To perform repetitive task with less code.

Types of loops:-

- for loop
- do/while
- for..in
- for--of.

Switch Statement

Switch statement :- It evaluates an expression compare its result with case values and execute the statement associated with the matching case.

Switch Syntax :-

```
Switch (expression) {
  case value1 :
       // body of case 1.
  break;
  case value 2 :
       // body of case 2
  break;
  default;
      // body of default
}
```

@programmer-girl--

break :- It is optional. It is used to end the ~~stat~~ switch statement.

Default :- If there is no matching case, the default body executes. It is optional.

For loop

For loop:- For loop executes a block of code as long as a specified condition is true.

Syntax:-

@programmer-girl--

```
for (initializer; condition ; iterator){
  {
      // statements
  }
```

→ Initializer:- It is an expression that initialize the loop, it executed once.

→ Condition:- It is a boolean expression that determines whether the for loop should execute or stop.

→ Iterator:- For statement executes the iterator after each iteration.

→ Example:-

```
for ( let i = 2 ; i < 4 ; i++) {
  console.log (i); }
```

While, Do While loops

While loop :- While loop executes statements as long as the conditions are true. If the condition become false, the loop is terminated.

Syntax :-

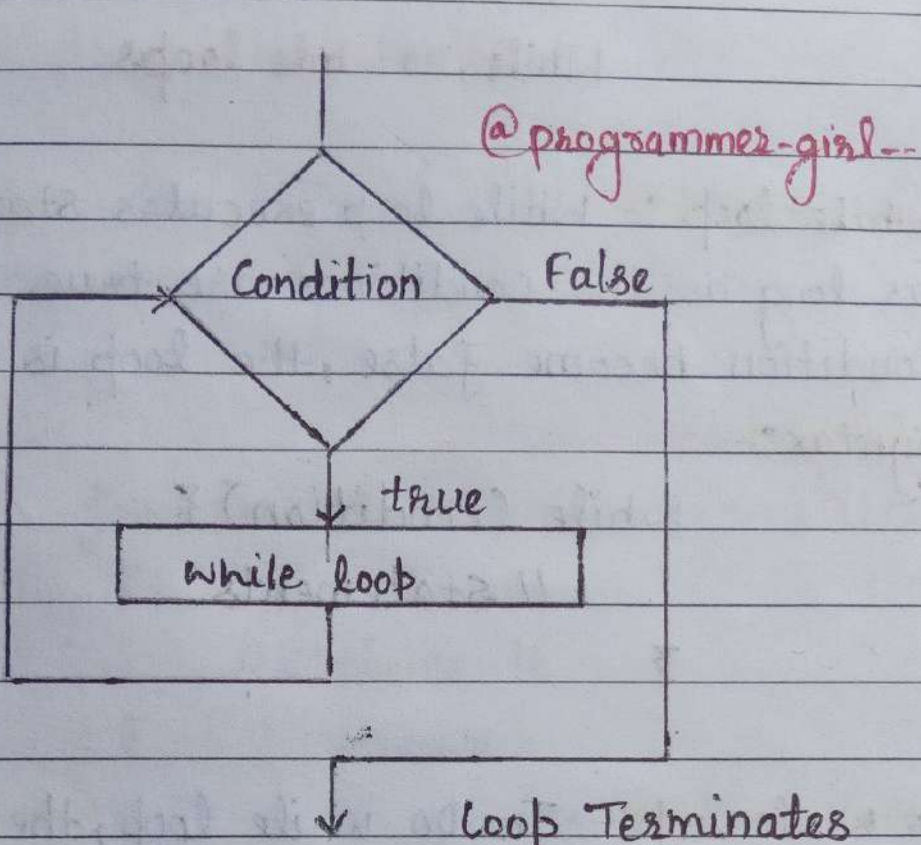
```
while (condition) {
    // statements
}.
```

Do While loop :- In Do while loop, the block of code executed once even before checking the condition.

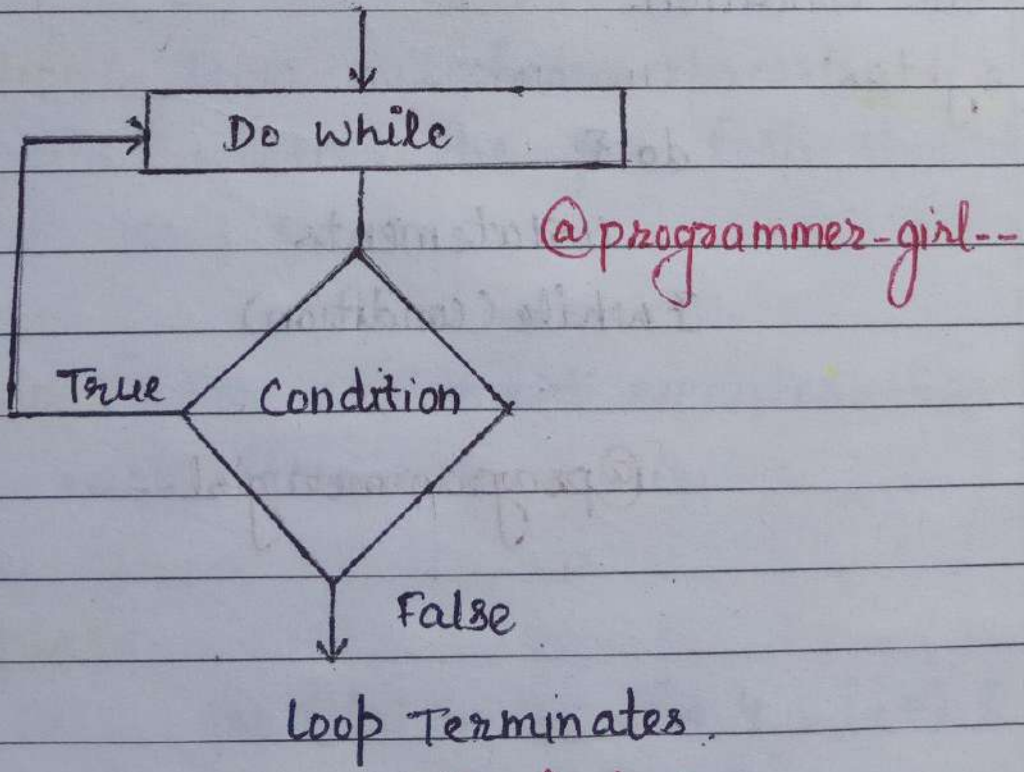
Syntax :-

```
do {
    //statements
} while (condition)
```

@programmer-girl--



-: While loop :-



-: Do... While loop :-

JavaScript Functions

Function:- A function is a block of code that performs a specific task.

Declare Function:-

```
function funName (){
    //statements
}
```

function is declared using the function keyword.

@programmer-girl--

Call Function:-

```
function funName(){
    //statements
}
funName (); → Call Function.
```

Example:-

```
function myName(){ → Declare Fun.
    console.log ("Smily");
}
myName (); → Function call
```

Output:- Smily

Advantage of function:-
Reusability
less code
Easy to understand

Function Parameters:- When we declare function we specify the parameters.

Function Arguments:- When we call function we specify the arguments.

For Example:-

```
Function example(Parameter){
console.log(Parameter);
}
let argument = 'arg';
example(argument);
```

@programmer-girl--

-: Intro to Arrays :-

Arrays:- It is a ordered collection of items.

```
                     Element / item
                          ↓
let pets = [ "cat", "dog", "cow" ];
     Index. →    0      1      2
```

JavaScript Array Characteristics

1 It can hold values of mixed types.
2. Size of Array is dynamic.

@programmer-girl--

```
let mixed = [ 1, 2.5, "cat" ]; → Mixed Type.
pets.push( "Monkey");   } → Dynamic Size.
Console.log(pets);      }
```

Accessing Array Elements

Arrays are zero-based indexed. It means the first element of array starts at index zero.

```
let pets = [ "cat", "dog" ];
Console.log(pets[0]); → Accessing element
```

output :- cat

-: Array Methods :-

1 **Array Length** :- It returns the number of elements in an array.

```
let num = [1, 2, 3, 4];
Console.log(num.length); // 4.
```

2. **Array Push()** :- It adds elements to the end of the array.

```
let num = [1,2,3];
Console.log( num.push(4));
// [1,2,3,4]
```

3. **Array Pop()** :- It removes the last element from an array and returns removed element.

```
let num = [1,2,3,4]
let removednum = num.pop();
Console.log(num); // [1,2,3]
Console.log (removednum); // 4.
```

@programmer-girl--

4. **Array shift() :-** It removes the first element and returns removed element from an array.

```
let num= [1,2,3,4];
let removednum = num.shift();
console.log(num); // [2,3,4]
console.log(removednum); // 1
```

@programmer-girl--

5. **Array unshift() :-** It adds elements to the beginning of an array.

```
let num = [1,2,3,4];
console.log(num.unshift(0));
// [0,1,2,3,4]
```

6. **Array Sort() :-** It sorts the items of an array.

```
Let num = [0,2,4,1];
Console.log (num.sort());
// [0,1,2,4]
```

7. **Array reverse() :-** It returns the reverse items of an array.

```
Let num = [1,2,3,4];
Console.log (num.reverse());
// [4,3,2,1]
```

Primitive vs Reference Types.

Primitive Types	Reference Types
It has a fixed size in memory.	Do not have a fixed size in memory.
Data stored on the stack	object stored in the heap.
Stored directly in the location.	Stored in the variable location is a pointer
For Example:- Null, String, Number, Bool undefined, Symbol	For Example:- Arrays, objects, Functions, Dates
we cannot add, delete update in primitive data.	we can add, delete update in reference data.

Spread Operator

Spread operator :- It is used to expand or spread an iterable or an array. It is denoted by three dots. (...)

For Example :-

```
let arrStr = [ 'A', 'B', 'C' ];
Console.log (arrStr);  // ['A', 'B', 'C'];
Console.log (... arrStr); // ABC
```

Clone Array using Spread operator :-

```
let arr1 = [1,2,3];
let arr2 = [... arr1]
Console.log(arr1);   // [1,2,3]
Console.log (arr2);  // [1,2,3]
// append an item to the array.
arr1.push (4);
Console.log (arr1);  // [1,2,3,4]
Console.log (arr2);  // [1,2,3]
```

@programmer-girl--

Array Destructuring

Array destructuring :- It is used to assign array values to distinct variables.

Example :-

```
const items = [ 'Books', 'Pen', 'Pencil' ];
const [x, y, z] = items → destructuring
Console. log(x);   // Books
Console. log(y);   // Pen
Console. log(z);   // Pencil
```

@programmer-girl--

Destructuring by using spread operator

```
const [x, ...y] = items
Console.log (x)   // Books
console.log (y)   // [ 'Pen', 'Pencil']
```

Note :- We should use variable with spread syntax as the last variable otherwis it throw error

I don't like error! Do you...

```
Const [...y, x]   // error
```

Objects Introduction

Objects:-

→ They are reference type
→ objects are good to handle real world data
→ objects stores data in key value pairs.
→ objects don't have index.

@programmer-girl--

Object declaration:-

```
        Const person = {
key  ← name: 'coder', → value
key  ← age: 20 → value
        };
        console.log(typeof person); // object
```

→ Access data from objects

```
Console.log(person.name);  // coder
Console.log(person.age);   // 20
                  ↳ dot notation
```

→ Add key-Value pair to objects

```
person.id = 5;
Console.log(person);
// {name: 'coder', age: 20, id: 5}
```

Anothes Method :- BracketNotation

```
Const person = {
        name : ' coder';
    "person age":- 22;
};
```

Key stored as a string by default.

Let's access data by Bracket Notation

```
console.log( person [ "name"]);
// output :- coder
```

Bracket Notation vs Dot Notation

In above example there is a key named as "person age" let's access it by dot Notation

Dot Notation:- `Console.log (person.person age);`

↳ It gives error because Is not include spaces between names.

Bracket Notation :- `Console.log ( person [" person age"]);`

↳ It works because it becomes string now.

@programmer-girl--

Iterate Object

-:- **Using for ... in loop**

```
let person = {
            firstname : 'programmer',
            last name : 'girl',
            age : 21                          @programmer-girl--
};
for ( let key in person) {
→       console.log (key); // output:- firstname
//It access                 //          lastname
        only key            //          age.

→       console.log (person[key]); // output:-
// It access                       // programmer
        only values                // girl
                                   // 21

→       console.log (key, " : ", person [key]);
It access        // output:- firstname : programmer
    both key-value.          lastname: girl
}                            age : 21
```

-:- Object.keys() :- This method was introduced in ES6. It takes an object and returns an array of the object properties. (key)

For Example:-

```
Console.log(object.keys(person));
```

// output:-

```
[ "firstname", "lastname", "age"]
```

Object.values() :- It takes an object and returns an array of the object values.

For Example :- @programmer-girl--

```
Console.log(object.values(person);
```

// output :-

```
["programmer", "girl", "21"]
```

Object.entries() :- It takes an object and returns the key-value pair.

For Example :-

```
console.log(object.enteries(person));
```

// output:- Try yourself---

Object Destructuring

Object destructuring:- It assigns properties of an object to individual variables.

Example:-

```
let person = {
        name: 'coder',
        age: 'twenty'
};
```

@programmer-girl--

```
let name = person.name;
let age = person.age;
```

We typically do like ↗

object destructuring

```
let {name, age} = person;
console.log(name);   // 'coder'
console.log(age);    // 'twenty'
```

Setting default values:-

```
let {name, age, class = ''} = person
console.log(class);   // ''
```

No class property in person object, Then we assign an empty string to the class.

Arrow Functions

Arrow Function :- Another way to write a function
It is Introduced in the ES6 version of JS.
It's Syntax is shorter than regular Function.

Example:-

$\nearrow$ Function Expression

```
let add = function (a,b){
        return a+b;
};
```

Above code using arrow function

```
let add = (x,y) => {
        return x+y;
};
```

Function with single parameter

```
(p1) => { Statements }    // Syntax.-1
 p1  => { Statements }    // Syntax-2.
```

Function with ~~no~~ parameter

```
let a = () => {    // Syntax.
    return 0
};
```

@programmer-girl--

Hoisting

Hoisting:- It is a behavior in which a function or a variable can be used before declaration.

Variable hoisting:-

```
console.log(name);  // undefined
var name = "xyz";
```

It doesn't cause an error. Because it looks like in execution phase:-

```
var name;   @programmer-girl--
console.log(name);
var name= "xyz";
```

In case of **let keyword**:-

```
console.log(name);  // Reference Error
let name = "xyz"
```

It cause an error. In case of let, variable is ~~not~~ hoisted but not initialized.

In case of **const keyword**:-

```
console.log(name);  // Error
const name = "xyz";
```

Conclusion:- let and const variables are hoisted but they cannot be accessed before their declaration.

Function Hoisting:- Function can be called before declaring it.

```
              name(); → Function called
Declaration ← function name() {
                  console.log('programmer-girl--');
              } → Formal function
          output :- programmer-girl--
```

Function Expression:- TypeError occurs in cas of function expression.

```
              name();      ↗ Function Expression.
              var name = function() {
              console.log('programmer-girl--');
              }
```

Arrow Function:-

```
              name();   // TypeError
              Var name = () => {
              console.log('programmer-girl--');
              }
```

Conclusion:- JavaScript doesn't hoist the function expressions and arrow functions.

@programmer-girl--

Lexical Scope

Lexical scope:- It means that a variable defined outside a function can be accessible inside another function defined after the variable declaration. But the opposite is not true.

For Example:-

```
function add(){
        var x = 4;  // y is not accessible
    function mul(){
          // x is accessible here, y is not
    function minus(){
         var y= 6;  // x is accessible.
    }
  }
}
```

@programmer-girl--

Note:- The variable defined inside a function will not be accessible outside the function.
In above code y is not accessible outside the function.

Block Scope vs Function Scope

Block Scope	Function Scope
It means that the variable is accessible within the block { }. @programmer-girl-.	It means that the variables are only accessible in the function in which they are declared.
let and const are block scope.	var is a function scope.

For Example:-
`for (let i=0; i<10; i++){`
`  // Block`
`}`
 	
`Console.log(i); // Error`
we are trying to access let variable outside the scope.
	For Example:-
	`function fun(){`
	`  var x = 42;`
	`}`
	`fun();`
	`Console.log(x); //Error`
	we cannot access var outside the function scope

Default Parameters

Default Parameters:- It allows us to give default values to the function parameters if no values is given.

For Example :-

Default parameters

```
function add(x=1, y=2){
    return x+y;
}
console.log(add(3,4));  // 7
console.log(add(5));    // x=5, y=2 ⇒ 7
console.log(add());     // x=1, y=2 ⇒ 3
```

Note:- A parameter has a default value is undefined

For Example:-

```
function add(x){
    console.log(x);
}
add(); // output:- undefined
```

@programmer-girl--

Rest Parameters:

Rest Parameters:- It is used to gather parameters and put them all in an array.

let's understand with an example:-

```
function test (a,b){
    console.log(a);  //8 → output
    console.log(b);  //9 → output
}
test(8;9);
test(8,9,7,6,5);
```

What will happen with 7,6,5 ? oops Nothing.
To cover 7,6,5 (Rest parameters concept comes)

```
                      → Rest parameter
                        Syntax.
function test (a, ...b){
    console.log(a);
    console.log(b);
}
test(8,9,7,6,5);
//output:- 8
            [9,7,6,5]
```

Hope you understand...

Parameter Destructuring

Parameter destructuring:- A couple of specific property values to pass as an parameter to the function defination, not in entire object.

For Example:-

```
Const user = {
    'name' : 'Alex',
    'age' : '20'          Param.
}                      ↗  destructurin
function userdetails ({name, age})
  {
     Console.log (name);
     Console.log (age);
  }
                 ⤻ user
userdetails( );
```

Output:-
```
Alex
20
```

@programmer-girl--

CallBack Function

Callback function :- It is defined as, we can also pass a function as an argument to a function.

For Example :-

```
function Second (name){
   console.log (name);
function first (callback){
   callback ('Alex');
 }
 first (Second); // output :- Alex.
```

Second function that is passed as an argument inside first function is called a callback function.

Note :- The callback function is helpful when you have to wait for a result that takes time.

@programmer-girl--

Sets in JavaScript

Sets :-

Stores a Collection of unique values
No index-based access
Order is not guaranted
Sets are iterable
Sets have its own methods.

Sets Syntax :-

```
let items = new Set();
```

typeof Sets :- object

@programmer-girl--

instanceof Sets :- True

Array vs Sets :-

- Array can have duplicate values whereas Sets cannot.
- In Array data is ordered by index whereas sets cannot

Useful Set methods.

```
const items = new Set();
```

- **add ( ) :-** Append a new element to the end of the set.

  For Example:-

  ```
  items.add ("Hi");
  // output :- Set (1) { 'Hi'}
  ```

- **clear ( ) :-** Remove all the elements and returns undefined.

  For Example:-

  ```
  const items = new Set ([1, 2, 3]);
  items.clear ();
  // output :- Set (0) {}
  ```

- **delete ( ) :-** It delete a specific element from Set.

  For Example:-

  ```
  const items = new Set ([1, 2, 3, 4]);
  items.delete (3);
  // output :- Set{3} {1, 2, 4}
  ```

- **has ( ) :-** check whether an element exists in Sets or not.

  ```
  items.has (2);   //output:- true.
  ```

map data structure

Map:- A Map is a collection of key-value pairs, similar to an object.

Map Syntax:-

```
const person = new Map();
```

★ Why we need map? If we have object
A Map is similar to object, keys in objects are only strings and symbols. But we can use any value as key In Map.

Let's Create a map:-

```
const person = new.Map();
person.set ('Name', "Alex");
 console. log (person);
```

```
//output:- {"Name" => "Alex"}
```

@programmer-girl--

Methods in Maps

1. **get():-** Returned the value associated with the key.

2. **set():-** Set the value of the key and returns the map.

3. **delete():-** Delete the entry which has the key same as passed key.

4. **clear():-** Delete all key-value pairs from the map.

5. **has():-** Returns true if the map has the key provided.

6. **keys():-** Returns the new iterator that containes the keys insertion order.

@programmer-girl-

Create Own Methods

Methods :- Function inside object is Known as methods.

For Example:-

```
const person= {
    name: "Alex",
    age: '21',
Method  ← about : function(){
        console.log( "My name is
              ${this.name} ");
    } }
person.about();
```

Output: My name is alex

when declaring a new object, use the object literal, not the new object() Constructor.

@programmer-girl..

This Keyword

this keyword :- "this" keyword refers to an object that is executing the current piece of code.

For Example:-

```
function Info() {
   console.log('my name is ${this.
      name}');
}
const person1 = {
      name: 'coder';
      about: Info
}
```

@programmer-girl--

```
const person2 = {
      name = 'girl';
      about: Info
}
person1.about(); → person1 name
person2.about(); → person2 name.
// output:-  my name is coder
           my name is girl.
```

Call, Apply, Bind Methods

- Call :- It invokes the function and allows you to pass in arguments one by one.
- Apply :- It invokes the function and allows you to pass in arguments as an array.
- Bind :- It returns a new function, allowing you to pass in a this array and any no. of arguments.

★ Call Example :-

```
        Const user1 = {
              name: 'Alex';
              age: 21,        @programmer-girl--
              intro: function(){
               console.log(this.name, this.age);
            }
     }

     Const user2 = {
              name: 'Jhon',    //output:-
              age: 31  }}  ↗    Jhon, 31
     user1.intro.call(user2);
```

★ **Apply Example :-**

```
var user1 = { name: 'Alex',
              age: 21 };
var user2 = { name: 'Jhon',
              age: 31 };
function say(greet){
console.log( greet + this.name);
}
say.apply(user2, ['Hi']);
// output:- HiJhon
```

★ **Bind Example:-**

```
var user1 = {name: 'Alex',
             age: 21 };
var user2 = { name: 'Jhon',
              age: 31 };
function say(){
   console.log(this.name, this.age);
}
var myfun = say.bind(user1);
myfun();
//output:- Alex 21
```

@programmer-girl--

Prototype

Prototype:- It is used to add new properties and methods to an existing object constructor.

@programmer-girl--

For Example:-

```
        function Person(){
            this.name= 'John',
             this. age = 23
        }
        Const person = new Person();
    Console.log(Person.prototype);
```
↙ checking the prototype value

```
// output:-   {...}
```

It shows an empty object.

Prototype Inheritance:- Objects inherit properties and methods from a prototype. Using the prototype makes faster object creation since properties / methods on the prototype don't have to be re-created each time a new object is created.

new keyword

Five things to remember about new keyword

1. It creates a new object. The type of this object is simply object

2. It sets this new object's internal.

3. It makes the this variable point to the newly created object

4. It executes the constructor function, using the newly created object whenever this is mentioned.

5. It returns the newly created object.

@programmer_girl--

Thank You

Keep learning!

Keep coding!

Keep Smiling :)

How do you feel about notes

-: tell us :-

@programmer-girl--