Loops:

Loops can execute a block of code as long as a specific condition is reached.

Loops are handy ( easy to use ) because they save time, reduce errors, and make code more readable.

While-> we don’t know the number of iterations.

Initialization; 🡨 int I = 0;

While ( condition ) { 🡨 ( i < 5 )  
// code

Increment/decrement 🡨 i++/i--

}

Do/while Loop: This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Initialization 🡨 int I = 0;

Do {

// code

Increment/decrement; 🡨 i++/ i--

} while( condition); 🡨 I < 5

For: When you know exactly how many times you want to loop through a block of code, use the for loop.

For ( initialization, condition, increment/decrement) {  
// code

}

* **For in loop:** The JavaScript for in statement loops through the properties of an Object:
  + for (key in object) {  
      // *code block to be executed*  
    }
  + const person = {fname:"John", lname:"Doe", age:25};  
      
    let text = "";  
    for (let x in person) {  
      text += person[x];  
    }
* The JavaScript for in statement can also loop over the properties of an Array:
  + for (variable in array) {  
      code  
    }
  + const numbers = [45, 4, 9, 16, 25];  
      
    let txt = "";  
    for (let x in numbers) {  
      txt += numbers[x];  
    }
* Do not use **for in** over an Array if the index **order** is important.
* { The way elements are stored or retrieved can vary depending on how the system is built. So, when using certain data structures like objects or maps, the order in which values appear might not match the order you added them. }
* **For of:** The JavaScript for of statement loops through the values of an iterable object.
* It lets you loop over iterable data structures such as Arrays, Strings, Maps, node lists, and more.
  + for (variable of iterable) {  
      // *code block to be executed*  
    }
* Loop over an array
  + const cars = ["BMW", "Volvo", "Mini"];  
      
    let text = "";  
    for (let x of cars) {  
      text += x;  
    }
* Loop over a string
  + let language = "JavaScript";  
      
    let text = "";  
    for (let x of language) {  
    text += x;  
    }

Objects:

In JavaScript, an object is a fundamental data structure used to store collections of related data and functionality.

It is a non-primitive data type that represents a collection of "properties," where each property consists of a key: value pair.

const car = {type: "Fiat", model: "500", color: "white"};

**Object Literal:** An object literal is a list of **key-value** pairs inside curly braces **{}**.

{firstName: "John", lastName: "Doe", age:50, eyeColor: "blue"}

**Creating a JavaScript Object: { create by 5 ways }**

1. **// Create an Object  
   const person = {  
     firstName: "John",  
     lastName: "Doe",  
     age: 50  
   };**

**And**

**// Create an empty js Object  
const person = {};  
  
// Add Properties  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";**

1. **// Create an Object  
   const person = new Object();  
     
   // Add Properties  
   person.firstName = "John";  
   person.lastName = "Doe";  
   person.age = 50;  
   person.eyeColor = "blue";**
2. **Class Animal {  
    constructor ( name ) {**

**This.name = name;**

**}**

**}**

**Const dog = new Animal(‘golden’);**

1. **Const john = Object.create(function);**

**Const function = {**

**Greet: function() {  
 // code**

**}**

**}**

* **Accessing Object Properties:**

***objectName.propertyName //* person.lastName;**

***objectName["propertyName"] //* person[“lastName”]**

* The object x is **not a copy** of a person. The object x **is** a person. The object x and the object person share the same memory address. Any changes to x will also change the person:

**//Create an Object  
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  age:50, eyeColor: "blue"  
}  
  
// Try to create a copy  
const x = person;  
  
// This will change the age in person:  
x.age = 10;**

**Adding New Properties to an object:**

* **const person = {**
* **firstname: "John",**
* **lastname: "Doe",**
* **age: 50**
* **};**
* **person.nationality = "English";**

**Delete Properties: The delete keyword deletes both the value of the property and the property itself. After deletion, the property cannot be used before it is added back again.**

* **const person = {**
* **firstname: "John",**
* **lastname: "Doe",**
* **age: 50**
* **};**
* **delete person.age; / delete person["age"];**

**Nested Objects:**

* **myObj = {  
    name: "John",  
    age:30,  
    myCars: {  
      car1: "Ford",  
      car2: "BMW",  
      car3: "Fiat"  
    }  
  }**
* **myObj["myCars"]["car2"];**
* **myObj.myCars["car2"];**
* **myObj.myCars.car2;**

**Methods in Objects: Object methods are actions that can be performed on objects.**

const **person** = {  
  firstName: "John",  
  lastName: "Doe",  
  id: 5566,  
  fullName: function() {  
    return **this**.firstName + " " + **this**.lastName;  
  }  
};

this refers to the **person object**, **this.firstName** means the **firstName** property of a **person** and lastName.

Accessing JS object:

* If you invoke the **fullName property** with (), it will execute as a **function**:
  + *objectName.methodName()*
* If you access the **fullName property** without (), it will return the **function definition**:
  + name = person.fullName;

Adding a new method to an object is easy:

* person.name = function () {  
    return (this.firstName + " " + this.lastName).toUpperCase();  
  };

How to Display JavaScript Objects:

Displaying a JavaScript object will output **[object Object]**.

* + // Create an Object  
    const person = {  
      name: "John",  
      age: 30,  
      city: "New York"  
    };  
      
    document.getElementById("demo").innerHTML = person;

Some solutions to display JavaScript objects are:

* Displaying the Object Properties by name
  + // Create an Object  
    const person = {  
      name: "John",  
      age: 30,  
      city: "New York"  
    };  
      
    // Display Properties  
    document.getElementById("demo").innerHTML =  
    person.name + "," + person.age + "," + person.city;
* Displaying the Object Properties in a Loop
  + // Create an Object  
    const person = {  
      name: "John",  
      age: 30,  
      city: "New York"  
    };  
      
    // Build a Text  
    let text = "";  
    for (let x in person) {  
      text += person[x] + " ";  
    };  
      
    // Display the Text  
    document.getElementById("demo").innerHTML = text;
  + You must use **person[x]** in the loop.
  + **person.x** will not work (Because **x** is the loop variable).
* Displaying the Object using Object.values(): creates an array from the property values:
  + // Create an Object  
    const person = {  
      name: "John",  
      age: 30,  
      city: "New York"  
    };  
      
    // Create an Array  
    const myArray = Object.values(person);  
      
    // Display the Array  
    document.getElementById("demo").innerHTML = myArray;
* Displaying the Object using JSON.stringify(): JavaScript objects can be converted to a string with JSON method JSON.stringify().
  + // Create an Object  
    const person = {  
      name: "John",  
      age: 30,  
      city: "New York"  
    };  
      
    // Stringify Object  
    let myString = JSON.stringify(person);  
      
    // Display String  
    document.getElementById("demo").innerHTML = myString;

**Object Constructor:**

To create an **object type** , we use an **object constructor function**. It is considered good practice to name constructor functions with an upper-case first letter.

function Person(first, last, age, eye) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eye;

this.nationality = "English"; 🡨 ( default properties )  
}

Now we can use the new Person() to create many new Person objects:

const myFather = new Person("John", "Doe", 50, "blue");  
const myMother = new Person("Sally", "Rally", 48, "green");  
const mySister = new Person("Anna", "Rally", 18, "green");  
const mySelf = new Person("Johnny", "Rally", 22, "green");

The new property will be added to **myFather**. Not to any other **Person Objects**.

myFather.nationality = "English";

To add a new property, outside the constructor, you must add it to the constructor function prototype:

Person.prototype.nationality = "English"; / Person.\_\_proto\_\_.nationality = “English”;

**Prototype Inheritance:**  
All JavaScript objects inherit properties and methods from a prototype:

* Date objects inherit from Date. prototype
* Array objects inherit from Array. prototype
* Person objects inherit from Person. prototype

The Object. The prototype is at the top of the prototype inheritance chain.

Adding Properties and Methods to Objects

function Person(first, last, age, eyecolor) {  
  this.firstName = first;  
  this.lastName = last;  
  this.age = age;  
  this.eyeColor = eyecolor;  
}  
  
Person.prototype.nationality = "English";

Person.prototype.name = function() {  
  return this.firstName + " " + this.lastName;  
};

You can only modify your prototypes. Never modify the prototypes of standard JavaScript objects.

**Object Methods:**

* **// Copies properties from a source object to a target object  
  Object.assign(target object, source object)  
    
  // Creates an object from an existing object  
  Object.create(object)  
    
  // Returns an array of the key/value pairs of an object  
  Object.entries(object)  
    
  // Creates an object from a list of keys/values  
  Object.fromEntries(object)  
    
  // Returns an array of the keys of an object  
  Object.keys(object)  
    
  // Returns an array of the property values of an object  
  Object.values(object)  
    
  // Groups object elements according to a function  
  Object.groupBy(object, callback)**

**Object Properties:**

**The Object.defineProperty() method:**

**Syntax - > Object.defineProperty(*object, property, descriptor*)**

**Add and change New Properties:**

**// Create an Object:  
const person = {  
  firstName: "John",  
  lastName : "Doe",  
  language : "EN"  
};  
  
// Add a Property  
Object.defineProperty(person, "year", {value:"2008"});**

**// Change a Property  
Object.defineProperty(person, "language", {value : "NO"});**

**JavaScript can secure better data quality when using getters and setters.**

**JavaScript Getter (The get Keyword):**

**// Create an object:  
const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "en",  
  get lang() {  
    return this.language;  
  }  
};  
  
// Display data from the object using a getter:  
document.getElementById("demo").innerHTML = person.lang;**

**JavaScript Setter (The set Keyword)**

**const person = {  
  firstName: "John",  
  lastName: "Doe",  
  language: "",  
  set lang(lang) {  
    this.language = lang;  
  }  
};  
  
// Set an object property using a setter:  
person.lang = "en";  
  
// Display data from the object:  
document.getElementById("demo").innerHTML = person.language;**

**Protection Methods:**

// Prevents re-assignment  
const car = {type: "Fiat", model: "500", color: "white"};  
  
// Prevents adding object properties  
Object.preventExtensions(object)  
  
// Returns true if properties can be added to an object  
Object.isExtensible(object)  
  
// Prevents adding and deleting object properties  
Object.seal(object)  
  
// Returns true if the object is sealed  
Object.isSealed(object)  
  
// Prevents any changes to an object  
Object.freeze(object)  
  
// Returns true if the object is frozen  
Object.isFrozen(object)