# **JavaScript**

## **Variables**: who store data temporarily in a computer’s memory. So, we store our data somewhere and give that memory location and name and with the name we can read it.

As we have been using **Var** keywords before Es6 to store variables.

But now **let** keyword is best way to declare a variable.

Rules to create variables:

* Cannot be a reserved keyword
* Should be meaningful
* Cannot start with a number
* Cannot contain a space or hyphen
* Together use camel notation (firstName) or pascal law (FirstName)
* Are case-sensitive(so the above firstName and FirstName are different)
* Multiple variables define
  + let firstName, lastName;
  + Best practice
    - let firstName
    - let lastName
* Assigning variables with const and changing later will make TypeError;
  + const stats = 4;
  + stats = 9
  + consol.log(stats);

## **Primitive / Value Types:** What kinds of values you can assign to variables

Strings: let name = ‘Body’; //string literal

Numbers: let age = 23; //number literal

Booleans: let isOpen = false; //Boolean literal

Undefined: let firstName = undefined; //not defined

Null: let lastName = null; //used to reassigned in Future : explicitly clear

## **Dynamic Typing:** Dynamic language like JavaScript type of variable can be changed at runtime.

Strings: let name = ‘Body’; //string literal

Numbers: let age = 23; //number literal

## **Reference Types:**

Object: let person = {name: ‘John’,age:30}

// let person = {}//Object literals

let person = {

    name: 'John',

    age: 30

}

//Dot notation

person.name = 'Billard';

//Bracket notation helps to

person['age'] = 33;

let selection = 'name'; //Sometime we don’t know the name of target property until the runtime.

person[selection] = 'Cooler';

console.log(person);

Array:

//Arrays

let selectedProducts = []; //Array literals; indicating empty array

selectedProducts = ['Chairs', 'Tables', 'dinning set'];

selectedProducts[3] = 'Shoes Racks'

console.log(selectedProducts.length);

Function:

// Function

//define function

function greet(name, lastName) {// name is parameter

    //body of function

    console.log(`Hello ${name} ${lastName}, Welcome!`);

}

//function call

greet('John', 'Coel'); //function with argument

greet('Marry', 'Moose');

## **Types of Functions**

// Function

//calculate a value

function square(number) {

    return number \* number;

}

let number = square(2);

console.log(number)

# **Object Oriented Programming**

## **What is Object-oriented Programming?** A programming paradigm or style of programming which is centred around objects rather functions.

## **Object-oriented principles**

4 Pillars of Object-oriented programming

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

### Encapsulation

Encapsulation is a bundling of the data and methods that operates on the data within one unit or class and hiding the values or state of an object inside the class. (Reduce complexity and increase reusability)

### Abstraction

It means that we should reduce the complexity by hiding unnecessary details in our class. (Reduce Complexity and isolate impact of changes)

### Inheritance

It is the mechanism for reusing the code (extends) i.e it allows you to eliminate redundant code. (Eliminate redundant code)

### Polymorphism

The ability of an object to take many forms. (Refactor ugly switch/case statements)

## **Objects**

1. Creating Objects
2. Factories and Constructors
3. Primitives and Reference Types
4. Working with Properties
5. Private Properties
6. Getters and Setters

### Creating Objects

//Object orient programming

//OOP concepts

//Objects

//Declaring variable using Var has issues like scoping so use either let or Const

//ES6

const circle = {//{}object literals syntax

    radius:1, //properties

    location: { //properties

        x: 1,

        y: 1

    },

    draw: function () { //methods

        console.log('draw');

    }

};

circle.draw();

### Factories and Constructors

* Factory function

//Declaring variable using Var has issues like scoping so use either let or Const

//ES6

//Factory function

function createCircle(radius) {

    return {//{}object literals syntax

        radius, //ES6-> radius: radius if key and value are same remove noise

        draw: function () { //methods

            console.log('draw');

        }

    };

}

//We can create a new object circle

const circle = createCircle(1); //calling factory function

circle.draw();

* Constructor function

//Constructor Function

function Circle(radius) {

    console.log("this", this)

    this.radius = radius;

    this.draw = function () {

        console.log('draw');

    }

}

const another = new Circle(1);

//when we use new operator to call a function

// it will create empty object ,(Circle {})

// Then it will set this to point to that object

// Finally it will return that object from this(Circle()) function

### Functions are object

A black screen with text

Description automatically generated

### Primitives and reference types

Encapsulation is a bundling of the data and methods that operates on the data within one unit or class and hiding the values or state of an object inside the class. (Reduce complexity and increase reusability)

### Working with properties

Encapsulation is a bundling of the data and methods that operates on the data within one unit or class and hiding the values or state of an object inside the class. (Reduce complexity and increase reusability)

### Private Properties

Encapsulation is a bundling of the data and methods that operates on the data within one unit or class and hiding the values or state of an object inside the class. (Reduce complexity and increase reusability)

### Getters and Setters

Encapsulation is a bundling of the data and methods that operates on the data within one unit or class and hiding the values or state of an object inside the class. (Reduce complexity and increase reusability