**Advanced and Object Oriented Javascript:**

JS is very popular bcoz it is a very flexible and dynamical language.

Javascript may not support class and packages but still JS is a Object Oriented language.

In JS, even functions are also considered as Objects.

Also in JS, the objects are very flexible. We can add or dlt the properties of object at any given point,

JS has two types,

1. Primitive type – Stores value directly in variable.
2. Reference type – As the name itself says, reference type refers/points to the memory location where the object is stored.

1.Primitive type:

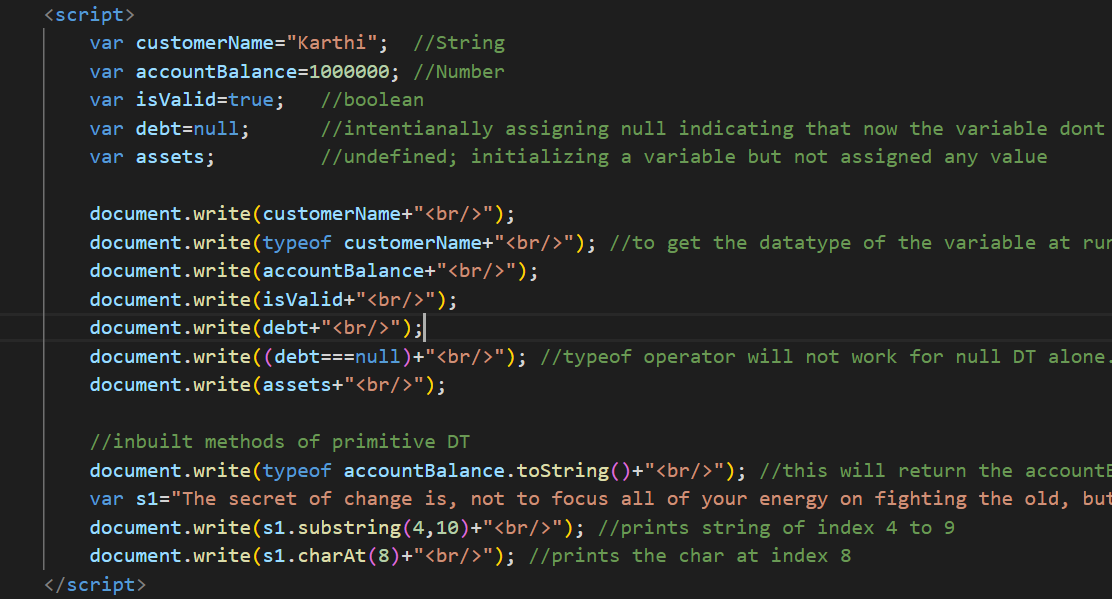
#Number – integer,float,decimal (Unlike java, JS only have Number DT, which comprises all these DT’s)

#String – Character, String

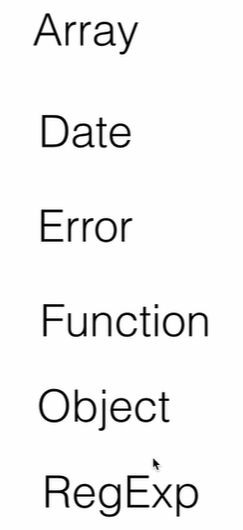
#Boolean – true, false

#Null – null(Represents an intentional absence of a value.Often used to indicate that a variable has been initialized but doesn't hold a value yet.)

#Undefined – when no value is assigned to a variable when declared.



**Reference Type:**

****

Reference type are the closest we can get to Class(Class in Java) in JS.

In JS, Objects are like the list/collection of Properties(Key&value).

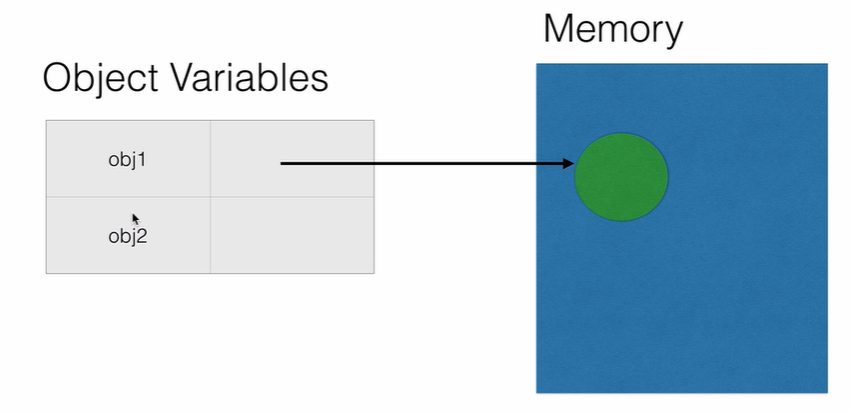
The key shld only be a string. The value can be any DT like Primitive DT value or reference DT(function,another object etc).

**Why the name Reference type:**

Object is a reference type bcoz unlike Primitive DT, values wont be stored directly.

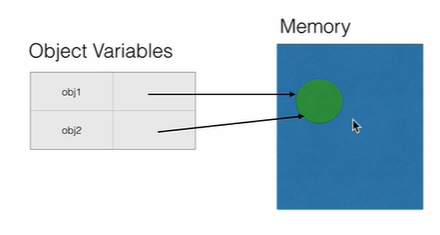
For example, when u create a obj, var obj1=new Object();

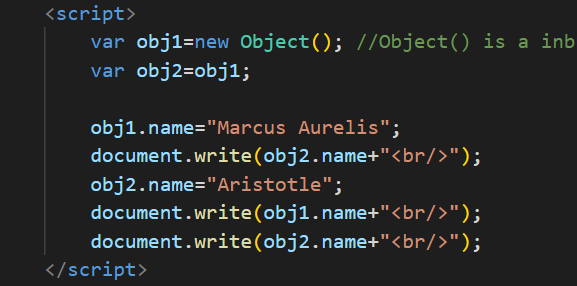
Here the Object() will not get directly assigned to the instance obj1. Object() will be stored in a memory location and the obj1 instance will point to that memory location.



To prove this, let us create another object instance obj2 and let us assign obj1 to obj2. Since we assigned obj1 to obj2, obj2 will also point to the same memory location where obj1 points to.

So both the object instances will have same properties and if we change the value of a property in one obj instance then the value of the other obj instance will also change automatically.





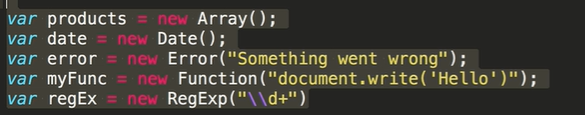
**Garbage Collection in JS:**

When you create a object in JS, JS engine will allocate a memory for that obj.

In C language, we have to deallocate the memory. But in java, it will automatically deallocate the memory when no obj reference points to that memory location.

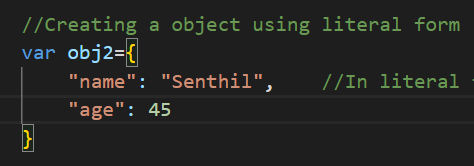
Same like in Java, in JS, the garbage collector will automatically deallocate the memory but for that we have to make sure to make the obj instance to null at the end of the program.

**Instantiating other Reference DT’s using ‘new’ keyword:**

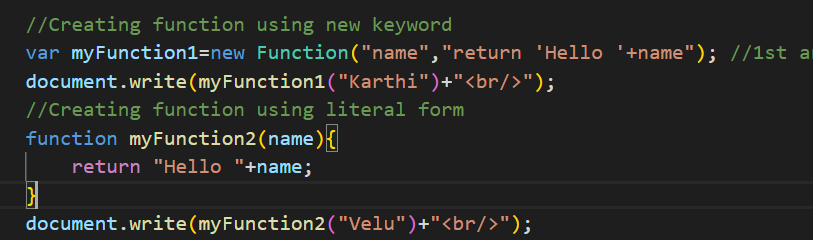


**Instantiating Reference DT’s using literal form:**

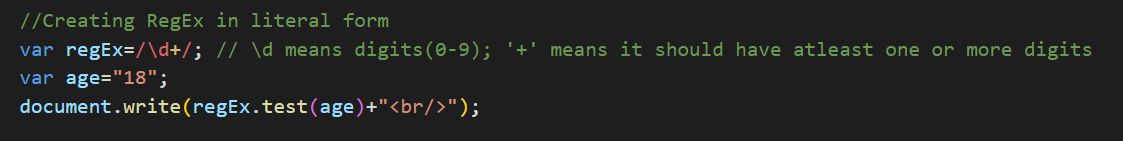
**1.Object:**

****

**2.Function:**

****

**3.RegEx:**

****

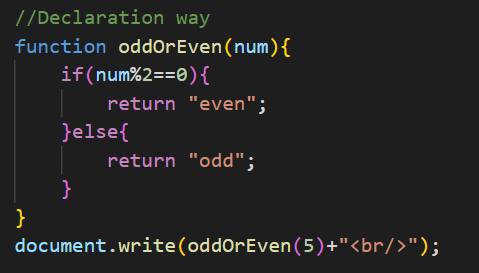
**Functions:**

In JS, functions are also objects. The only diff btwn a function and object is we can call the function.

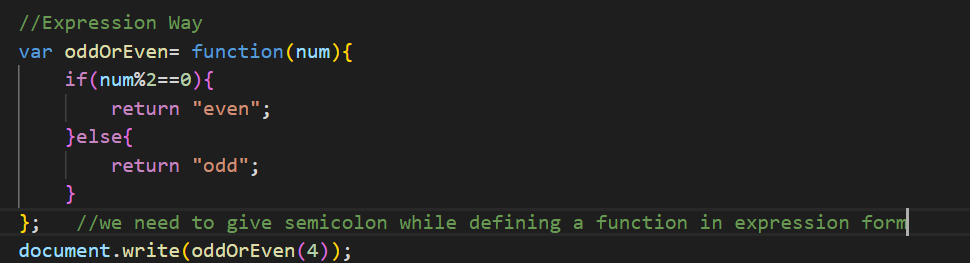
Also we can add or remove any no of parameters in function dynamically in JS.

**Two ways of defining a function:**

1.Declaration way:



2.Expression way(also known as Anonymous way):



**Function Hoisting(like flag hoisting):**

In Function Declaration, even if u call the function before the function definition, it works.

In Function Expression, if u call the function before the function definition, it will not work as the function name is given in a variable.

So, according to our needs, we have to choose between 2 ways of function definition.

**Variable arguments:**

In JS, even though a function has only 2 parameters, while calling that function, we can pass any no of parameters. But the function will accept only the first 2 parameters and pass it but we can access and use the rest of parameters passed, by using 'arguments' variable.

‘arguments’ is like a kind of data structure or an array which every function in JS has. It contains all the parameters passed in function calling so that we can get and use all the parameters passed while the function is called.

A screen shot of a computer program

Description automatically generated

**Function Overloading problem in JS:**

Function overloading means functions with same name but diff no of parameter or parameter type.

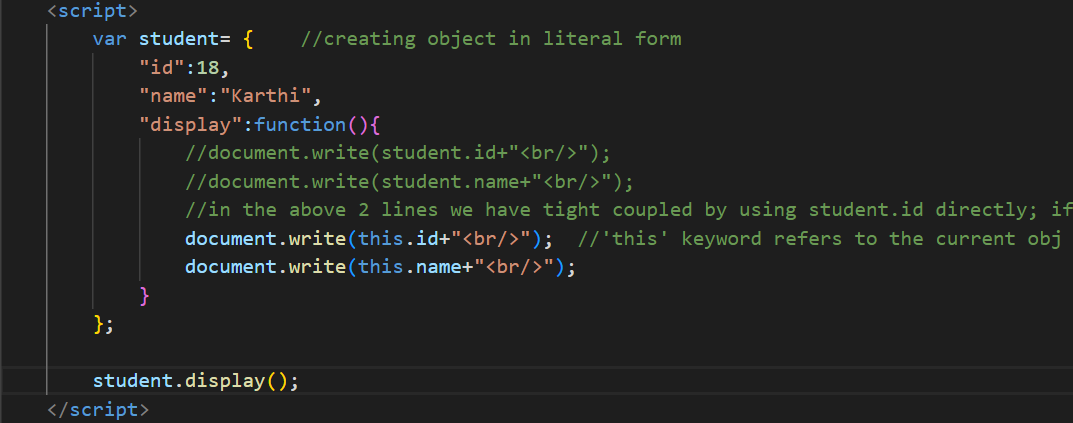
Since in JS, passing the parameters in function calling is dynamic(we can pass any no of parameters), JS will confuse which function to call and always call the last defined function.

To avoid this problem, we can use arguments variable; Define only one function.

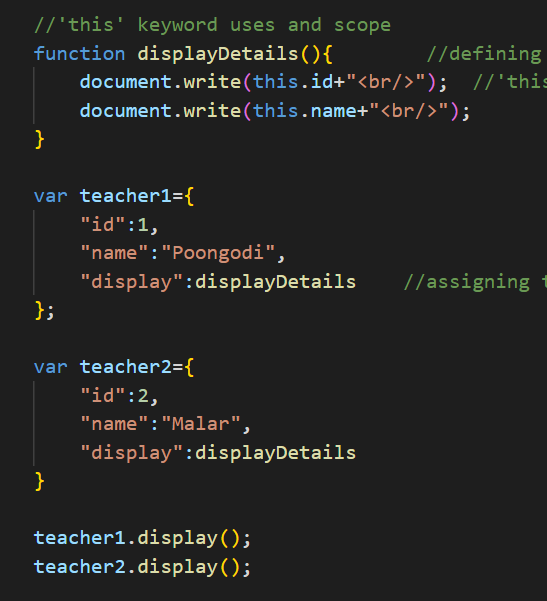
A screen shot of a computer code

Description automatically generated

**Creating function inside Object:**



‘this’ keyword uses and scope:



'this' keyword refers to the current obj; if teacher1 calls displayDetails() method, 'this' keyword  acts as teacher1 and same like for teacher2.

**call():**

**apply():**

**bind():**

**Objects:**

Unlike other OOP languages, objects in JS are dynamic. Because we can add, remove, modify the properties of the object at any time.

**‘in’ operator:**

**hasOwnProperty():**

**Deleting properties from object:**

**Retrieving properties from object using ‘in’ operator:**

**Retrieving properties from object using inbuilt generic Object:**

**Enumerability check:**

**Note:**

**Accessor Properties:**

**Object Property Attributes:**

**A screen shot of a computer code

Description automatically generated**

**Data Property Attributes:**

When the configurable attribute of a property is set to false, it imposes certain restrictions on what can be done with that property. Here’s a detailed explanation:

Restrictions When configurable is false

1. Cannot Delete the Property:
   * The property cannot be removed from the object using the delete operator.
2. Cannot Redefine the Property:
   * You cannot change the property’s attributes (other than writable).
   * You cannot convert the property from a data property to an accessor property, or vice versa.

What You Can Still Do

* Change the Value:
  + If the property is a data property and its writable attribute is true, you can still change its value.

Example

Here’s an example to illustrate these points:

JavaScript

var person = {};

// Define a non-configurable property

Object.defineProperty(person, 'name', {

value: 'Mark Antony',

writable: true,

enumerable: true,

configurable: false

});

console.log(person.name); // "Mark Antony"

// Attempt to delete the property

delete person.name; // false (property is not deleted)

console.log(person.name); // "Mark Antony"

// Attempt to redefine the property

try {

Object.defineProperty(person, 'name', {

value: 'Baasha'

});

} catch (e) {

console.log(e.message); // Cannot redefine property: name

}

// Change the value (since writable is true)

person.name = 'Baasha';

console.log(person.name); // "Baasha"

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

In this example:

* The name property is defined with configurable: false.
* Attempting to delete the name property fails.
* Attempting to redefine the name property results in an error.
* Changing the value of name is allowed because writable is true.

Summary

* Configurable: false prevents deletion and redefinition of the property.
* You can still change the value if the property is writable

**Accessor Property Attributes:**

**Defining Multiple Properties:**

**Retrieving Attribute values of property:**

**Preventing Extensions using preventExtensions(), seal() and freeze():**

**Constructors and Prototypes:**

**Constructors:**

**Prototypes:**

**A close-up of a text

Description automatically generated**

Every function has a inbuilt property called prototype. Prototype is mainly used for memory management when multiple instances are created for a object and also for inheritance.

**Custom Prototype:**

**Constructor property problem of Prototype when using literal form:**

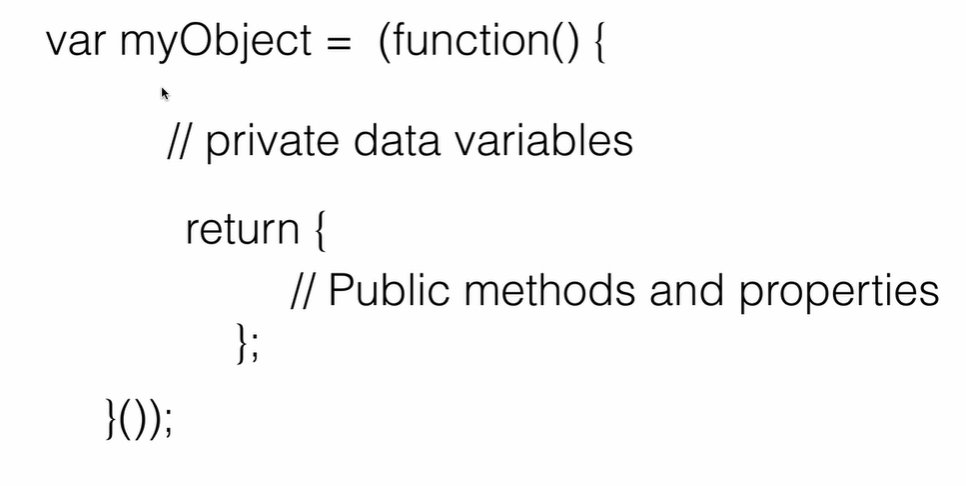
**Using prototype on inbuilt objects:**

**Inheritance(Includes Object.create()):**

**Constructor Inheritance:**

**Using variables and functions of parent obj in child obj(super keyword):**

**Module Pattern:**

****

**Revealing Module Pattern:**

**ES6(ECMAScript 6):**

**Scope(Balachandra - Youtube):**

**Var:**

**Let:**

**Const:**

**Global and Local Scope:**

**Template Strings and backtick:**

**Class:**

**Inheritance and Encapsulation in Class:**

**Arrow Function(Thippireddy Udemy):**

**Destructuring:**

**Object Destructuring:**

**Array Destructuring:**

**Function Destructuring:**

**For-of loop:**

**Map:**

**Looping Map:**

**Set:**

**Promises:**

**Real world App use of Promises:**

In a real-world application, you would replace the setTimeout with an actual API call using fetch or another HTTP client library:

function fetchUserData(userId) {

return fetch(`https://api.example.com/users/${userId}`)

.then(response => {

if (!response.ok) {

throw new Error("Network response was not ok");

}

return response.json();

});

}

fetchUserData(1)

.then(userData => {

console.log("User Data:", userData);

document.getElementById("userProfile").innerText = `Name: ${userData.name}, Age: ${userData.age}`;

})

.catch(error => {

console.error(error);

document.getElementById("userProfile").innerText = "Error fetching user data";

});

Using Promises allows you to handle asynchronous operations like fetching data from an API in a clean and manageable way. It helps you deal with the success and failure of these operations without getting into callback hell.

**Async-Await:**

**DOM(Youtube – Balachandra)(Refer in JavaScript Fundamentals folder):**

**DOMmethods:**

**DOMmethods2:**