==========Objects==========

*objects are used to store keyed collections of various data and more complex entities.
*An object can be created with figure brackets {…} with an optional list of properties.
  =>A property is a “key: value” pair, where key is a string (also called a “property name”), and value can be anything.

An empty object (“empty cabinet”) can be created using one of two syntaxes:
---

synatx1: let user = new Object(); // "object constructor" syntax

syntax2: let user = {}; // "object literal" syntax

Literals and properties:

We can immediately put some properties into {...} as “key: value” pairs:
---

ex:

```
let user = {     // an object
  name: "sathya",  // by key "name" store value "sathya"
  age: 30        // by key "age" store value 30
};
```

**A property has a key (also known as “name” or “identifier”) before the colon ":" and a value to the right of it.

In the user object, there are two properties:

The first property has the name "name" and the value "sathya".
The second one has the name "age" and the value 30.

Property values are accessible using the dot notation:

```
// get property values of the object:
alert( user.name ); // sathya
alert( user.age ); // 30
```

*if its boolean type: user.isAdmin = true;

=>To remove a property, we can use the delete operator:

synatx: delete user.age;

ex:

```
let user = {
  name: "sathya",
  age: 30
};

let key = prompt("What do you want to know about the user?", "name");

// access by variable
alert( user[key] ); // John (if enter "name")
```

we cant do same by using dot notation:
----------***

```
let user = {
  name: "John",
  age: 30
};

let key = "name";
alert( user.key ) // undefined
```

Computed properties:

We can use square brackets in an object literal, when creating an object. That’s called computed properties.

ex:

```
let fruit = prompt("Which fruit to buy?", "apple");
let bag = {};

// take property name from the fruit variable
bag[fruit] = 5;
```

ex:

```
function makeUser(name, age) {
  return {
```

```
    name: name,
    age: age,
    // ...other properties
  };
}

let user = makeUser("sathya", 30);
alert(user.name); // sathya
```

Property existence test, “in” operator:

---

ex:

```
let user = { name: "sathya", age: 30 };

alert( "age" in user ); // true, user.age exists
alert( "blabla" in user ); // false, user.blabla doesn't exist
```

ex:

```
let user = { age: 30 };

let key = "age";
alert( key in user ); // true, property "age" exists
```

The "for..in" loop:

The syntax:

```
for (key in object) {
  // executes the body for each key among object properties
}
```

ex:

--

```
let codes = {
  "49": "Germany",
  "41": "Switzerland",
  "44": "Great Britain",
  // ..,
  "1": "USA"
```

```
};

for (let code in codes) {
  alert(code); // 1, 41, 44, 49
}


ex:
---

let user = {
  name: "John",
  surname: "Smith"
};
user.age = 25; // add one more

// non-integer properties are listed in the creation order
for (let prop in user) {
  alert( prop ); // name, surname, age
}



Objects are associative arrays with several special features.

*They store properties (key-value pairs), where:

1.Property keys must be strings or symbols (usually strings).
2.Values can be of any type.
3.To access a property, we can use:

*The dot notation: obj.property.
*Square brackets notation obj["property"]. Square brackets allow taking the key from
a variable, like obj[varWithKey].
Additional operators:
-----------------

1.To delete a property: delete obj.prop.
2,To check if a property with the given key exists: "key" in obj.
3.To iterate over an object: for (let key in obj) loop.
4.What we've studied in this chapter is called a "plain object", or just Object.

There are many other kinds of objects in JavaScript:
==========================================
*Array to store ordered data collections,
*Date to store the information about the date and time,
*Error to store the information about an error.


Object references and copying:
```

==========

One of the fundamental differences of objects versus primitives is that objects are stored and copied “by reference”,

==>whereas primitive values: strings, numbers, booleans, etc – are always copied “as a whole value”.

ex:

Here we put a copy of message into phrase:

```
let message = "Hello!";
let phrase = message;
```

Objects are not like that.*****

***A variable assigned to an object stores not the object itself, but its “address in memory” – in other words “a reference” to it.

```
        ex;
        --
        let user = {
  name: "keerthi"
};
```

**When an object variable is copied, the reference is copied, but the object itself is not duplicated.

```
 ex:
 let user = { name: "keerthi" };

let admin = user; // copy the reference
```

ex:

```
let user = { name: 'John' };

let admin = user;

admin.name = 'tarun'; // changed by the "admin" reference

alert(user.name); // 'tarun', changes are seen from the "user" reference
```

Comparison by reference:
---

*Two objects are equal only if they are the same object.

ex:

```
let a = {};
let b = a; // copy the reference

alert( a == b ); // true, both variables reference the same object
alert( a === b ); // true
```

**both objects are indpendent

ex:
--

```
let a = {};
let b = {}; // two independent objects

alert( a == b ); // false
```

Cloning and merging, Object.assign:====
==========

*opying an object variable creates one more reference to the same object.

*We can create a new object and replicate the structure of the existing one,
by iterating over its properties and copying them on the primitive level.

ex:

```
let user = {
  name: "John",
  age: 30
};

let clone = {}; // the new empty object

// let's copy all user properties into it
for (let key in user) {
```

```
  clone[key] = user[key];
}

// now clone is a fully independent object with the same content
clone.name = "Pete"; // changed the data in it

alert( user.name ); // still John in the original object
```