


JavaScript Fundamentals

Instructor: Bui Binh Giang
buibinhgiang@vanlanguni.vn


JavaScript

**JAVASCRIPT IS A HIGH-LEVEL, OBJECT-ORIENTED,
MULTI-PARADIGM PROGRAMMING LANGUAGE.**


We don't have to worry about complex stuff like memory management



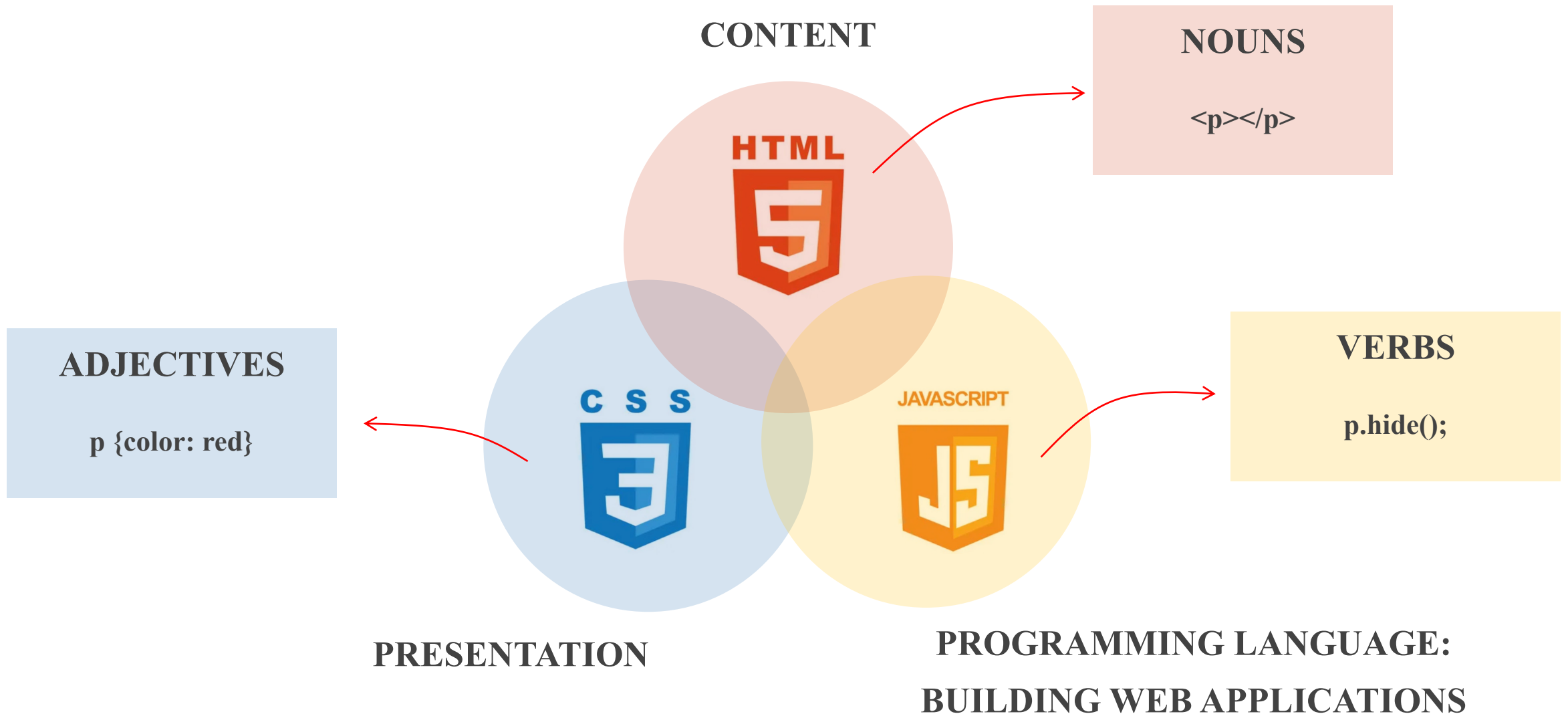
Base on object for storing most kind of data



We can use different style of programming



The role of JavaScript in Web development



You can do almost thing with JavaScript

Dynamic effects and web applications

JS



Back-end applications

JS



Native mobile applications

JS

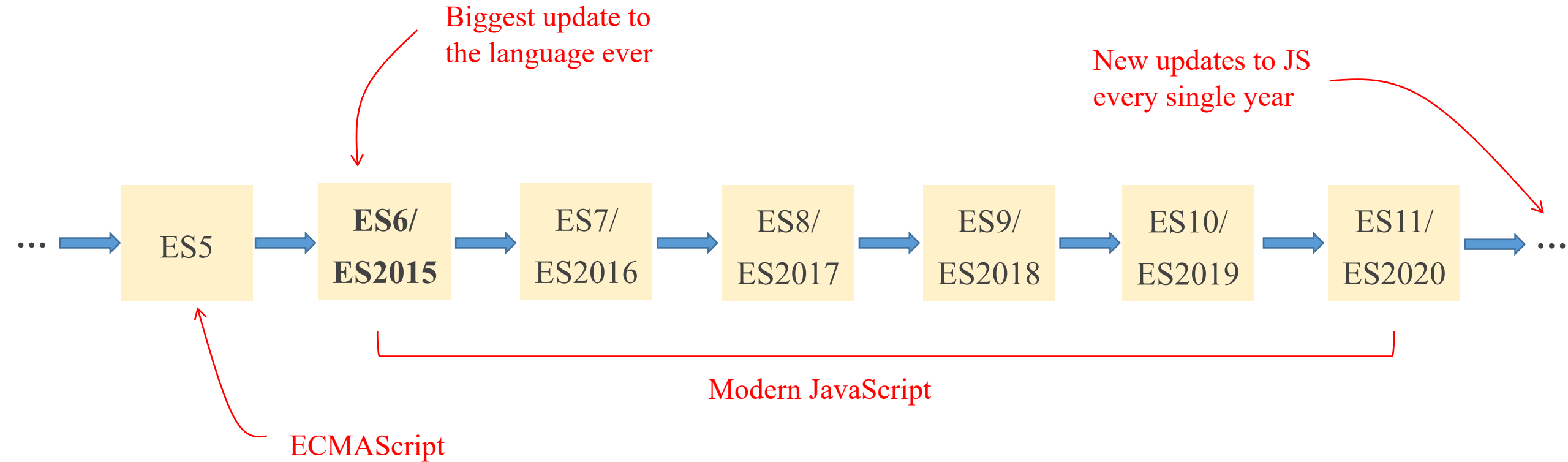


Native desktop applications

JS



JavaScript Releases...

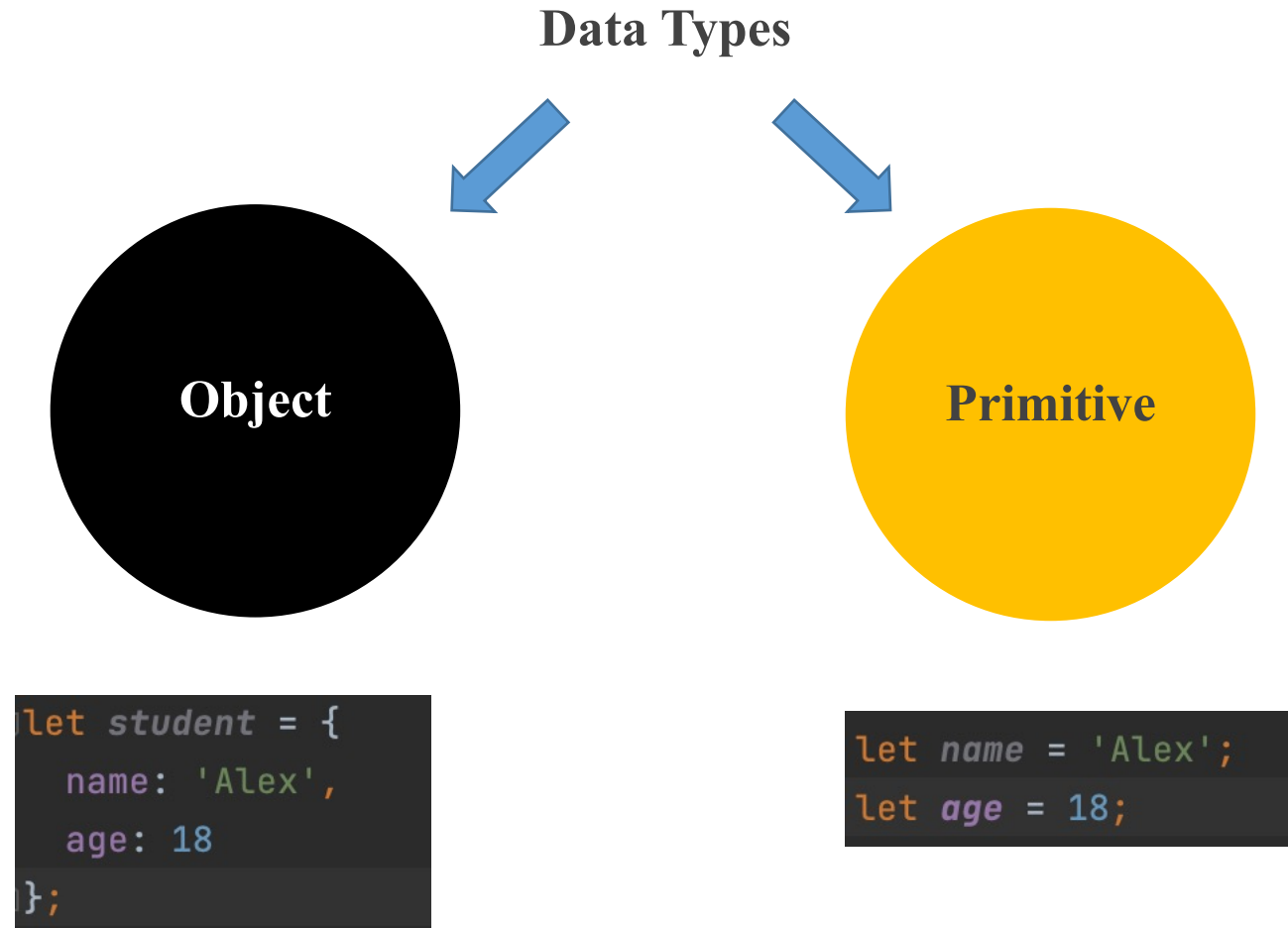


JavaScript

JavaScript consists of three main parts:

- ECMAScript provides the core functionality.
- The **Document Object Model (DOM)** provides interfaces for interacting with elements on web pages
- The **Browser Object Model (BOM)** provides the browser API for interacting with the web browser.

Data Types



DATA TYPE - PRIMITIVE

Primitive data types

- **Number:** Floating point numbers → User for decimals and intergers.
- **String:** Sequence of characters → Used for text
- **Boolean:** Logical type that can only be true or false → Used for taking decisions.
- **Undefined:** Value take by a variable that is not yet defined (“empty value”)
- **Null:** Also means “empty value”
- **Symbol (ES2015):** Value that is unique and cannot be changed
- **BigInt (ES2020):** Larger integers than the Number type can hold

Boolean

Type	true	false
String	Non-empty string	Empty string
Number	Non-zero number and Infinity	0, NaN
Object	Non-null object	null
Undefined		undefined

JavaScript Variables

By default, the variable has a special value **undefined** if you have not assigned a value to it.

You don't need to specify the variable's type in the declaration like other static typed languages such as Java or C#

Variable names follow these rules:

- Variable names are **case-sensitive** → “message” and ”Message” are different variables.
- Variable names can only contain letters, numbers, underscores, or dollar signs and cannot contain spaces. Also, variable names must begin with a letter, an underscore (`_`) or a dollar sign (`$`)
- Variable names cannot use the **reserved words**.

JavaScript Variables

Declare a variable:

var message;

let content;

Initialize a variable:

let message;

message = "hello";

let content = "hello";

let name = "Alex", age = 18;

Constants

A constant holds a value that doesn't change:

```
const workday = 5;
```

```
const pi = 3.14;
```

```
...
```

Reserved words

Reserved words defined in ECMA-262:

break, case, catch, continue, debugger, default, else, export, extends, function, if, import, new, return, super, throw, try, null, void, while, with, class, delete, finally, in, switch, typeof, yield, const, do, for, instanceof, this, var

In addition to the reserved keywords, ECMA-252 also define a list of future reserved words that cannot be used as identifiers or property names:

enum, implements, let, protected, private, public, await, interface, package, implements, public

DATA TYPE - OBJECT

JavaScript Objects

- An object is an unordered collection of key-value pairs. Each **key-value** pair is called a property.
- The key of a property can be a string. And the value of a property can be any value
- You can add a property to an object after object creation
- You can delete a property using “**delete**” operator
- To check if a property exists in an object, you use the “**in**” operator

JavaScript Objects

Creating an Object:

```
let empty = {};
```

```
let person = {  
  firstName: 'John',  
  lastName: 'Doe'  
};
```

Accessing properties:

The dot notation (.)

```
objectName.propertyName
```

Array-like notation ([])

```
objectName[ 'propertyName' ]
```

JavaScript Objects

Deleting a property of an object:

```
delete objectName.propertyName;
```

Checking if a property exists:

```
propertyName in objectName
```

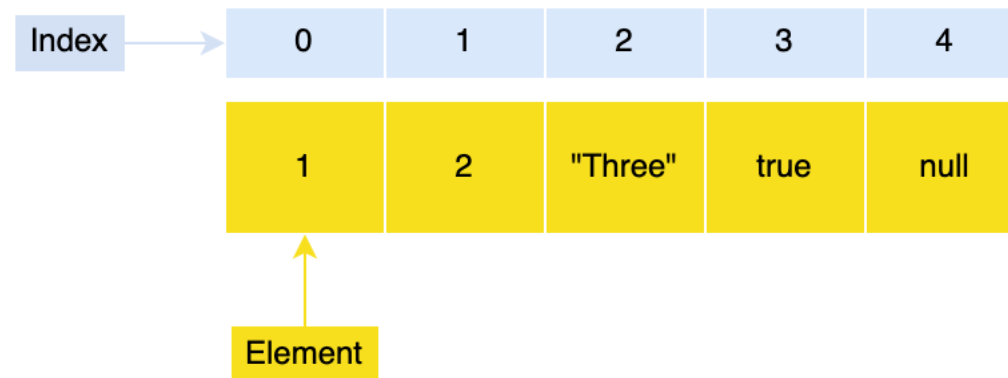
Exercise - 101

Requirement:

- Create an object that describe a Person with following properties: full name, age, height, weight.
- Replace property “full name” with following properties: first name, last name.

JavaScript Arrays

- In JavaScript, an array is an ordered list of values. Each value is called an element specified by an index.
- An array can hold values of mixed types.
- JavaScript arrays are dynamic, which means that they grow or shrink as needed.



JavaScript Arrays

Creating JavaScript arrays:

```
let scores = new Array();
```

```
let scores = new Array(9,10,8,7,6);
```

```
let scores = Array(10);
```

```
let colors = ['red', 'green', 'blue'];
```

Accessing JavaScript array elements:

```
arrayName[index]
```

```
let mountains = ['Everest', 'Fuji', 'Nanga Parbat'];
```

```
console.log(mountains[0]); // 'Everest'
```

```
console.log(mountains[1]); // 'Fuji'
```

```
console.log(mountains[2]); // 'Nanga Parbat'
```

JavaScript Arrays

Getting the array size:

```
let mountains = ['Everest', 'Fuji', 'Nanga Parbat'];  
console.log(mountains.length); // 3
```

Adding an element to the end of an array:

```
let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];  
seas.push('Red Sea');
```

Adding an element to the beginning of an array:

```
let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];  
seas.unshift('Red Sea');
```

JavaScript Arrays

Removing an element from the end of an array:

```
let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];  
const lastElement = seas.pop();
```

Removing an element from the beginning of an array:

```
let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];  
const firstElement = seas.shift();
```

Finding an index of an element in the array:

```
let seas = ['Black Sea', 'Caribbean Sea', 'North Sea', 'Baltic Sea'];  
let index = seas.indexOf('North Sea');  
  
console.log(index); // 2
```

ARITHMETIC OPERATORS

JavaScript Arithmetic Operators

Operator	Sign
Addition	+
Subtraction	-
Multiplication	*
Division	/

JavaScript Arithmetic Operators

Addition operator (+)

- The addition operator returns the sum of two values.
- If both values are strings, it concatenates the second string to the first one.
- If one value is a string, it implicitly converts the numeric value into a string and concatenates two strings

JavaScript Arithmetic Operators

Addition operator (+) with special numbers

First Value	Second Value	Result
NaN		NaN
Infinity	Infinity	Infinity
-Infinity	-Infinity	-Infinity
Infinity	-Infinity	NaN

JavaScript Arithmetic Operators

Subtraction operator (-)

- The subtraction operator (-) subtracts one number from another.
- If either value is not a number, the JavaScript engine implicitly converts it into a number using the `Number()` function and performs the Subtraction.

JavaScript Arithmetic Operators

Subtraction operator (-) with special numbers

First Value	Second Value	Result
NaN		NaN
Infinity	Infinity	NaN
-Infinity	-Infinity	NaN
Infinity	-Infinity	Infinity

JavaScript Arithmetic Operators

Multiplication operator (*)

- The multiplication operator multiplies two numbers and returns a single value.
- If either value is not a number, the JavaScript engine implicitly converts it into a number using the `Number()` function and performs the multiplication.

JavaScript Arithmetic Operators

Multiplication operator (*) with special numbers

First Value	Second Value	Result
NaN		NaN
Infinity	0	NaN
Infinity	Infinity	Infinity
-Infinity	-Infinity	Infinity
Infinity	-Infinity	-Infinity
Infinity	Positive number	Infinity
Infinity	Negative number	-Infinity

JavaScript Arithmetic Operators

Divide operator (*)

- The divide operator divides the first value by the second one.
- If either value is not a number, the JavaScript engine implicitly converts it into a number using the `Number()` function and performs the division.

JavaScript Arithmetic Operators

Divide operator (*) with special numbers

First Value	Second Value	Result
NaN		NaN
A number	0	Infinity
Infinity	Infinity	NaN
0	0	NaN
Infinity	Positive number	Infinity
Infinity	Negative number	-Infinity

JavaScript Arithmetic Operators

Using JavaScript arithmetic operators with objects:

- If a value is an object, the JavaScript engine will call the **valueOf()** method of the object to get the value for calculation

```
let energy = {  
  valueOf() {  
    return 100;  
  },  
};  
  
let currentEnergy = energy - 10;  
console.log(currentEnergy);  
  
currentEnergy = energy + 100;  
console.log(currentEnergy);  
  
currentEnergy = energy / 2;  
console.log(currentEnergy);  
  
currentEnergy = energy * 1.5;  
console.log(currentEnergy);
```

JavaScript Arithmetic Operators

Using JavaScript arithmetic operators with objects:

- If the object doesn't have the **valueOf()** method but has the **toString()** method, the JavaScript engine will call the **toString()** method to get the value for calculation

```
let energy = {  
  toString() {  
    return 50;  
  },  
};  
  
let currentEnergy = energy - 10;  
console.log(currentEnergy);  
  
currentEnergy = energy + 100;  
console.log(currentEnergy);  
  
currentEnergy = energy / 2;  
console.log(currentEnergy);  
  
currentEnergy = energy * 1.5;  
console.log(currentEnergy);
```

JavaScript Assignment Operators (=)

Operator	Meaning	Description
<code>a = b</code>	<code>a = b</code>	Assigns the value of <code>b</code> to <code>a</code> .
<code>a += b</code>	<code>a = a + b</code>	Assigns the result of <code>a</code> plus <code>b</code> to <code>a</code> .
<code>a -= b</code>	<code>a = a - b</code>	Assigns the result of <code>a</code> minus <code>b</code> to <code>a</code> .
<code>a *= b</code>	<code>a = a * b</code>	Assigns the result of <code>a</code> times <code>b</code> to <code>a</code> .
<code>a /= b</code>	<code>a = a / b</code>	Assigns the result of <code>a</code> divided by <code>b</code> to <code>a</code> .
<code>a %= b</code>	<code>a = a % b</code>	Assigns the result of <code>a</code> modulo <code>b</code> to <code>a</code> .
<code>a &= b</code>	<code>a = a & b</code>	Assigns the result of <code>a</code> AND <code>b</code> to <code>a</code> .
<code>a = b</code>	<code>a = a b</code>	Assigns the result of <code>a</code> OR <code>b</code> to <code>a</code> .
<code>a ^= b</code>	<code>a = a ^ b</code>	Assigns the result of <code>a</code> XOR <code>b</code> to <code>a</code> .
<code>a <<= b</code>	<code>a = a << b</code>	Assigns the result of <code>a</code> shifted left by <code>b</code> to <code>a</code> .
<code>a >>= b</code>	<code>a = a >> b</code>	Assigns the result of <code>a</code> shifted right (sign preserved) by <code>b</code> to <code>a</code> .
<code>a >>>= b</code>	<code>a = a >>> b</code>	Assigns the result of <code>a</code> shifted right by <code>b</code> to <code>a</code> .

JavaScript Unary Operators

Unary Operators	Name	Meaning
+x	Unary Plus	Convert a value into a number
-x	Unary Minus	Convert a value into a number and negate it
++x	Increment Operator (Prefix)	Add one to the value
--x	Decrement Operator (Prefix)	Subtract one from the value
x++	Increment Operator (Postfix)	Add one to the value
x--	Decrement Operator (Postfix)	Subtract one from the value

JavaScript Unary Operators

- Unary operators work on one value.
- Unary plus (+) or minus (-) converts a non-numeric value into a number. The unary minus negates the value after the conversion.
- The prefix increment operator adds one to a value. The value is changed before the statement is evaluated.
- The postfix increment operator adds one to a value. The value is changed after the statement is evaluated.
- The prefix decrement operator subtracts one from a value. The value is changed before the statement is evaluated.
- The postfix decrement operator subtracts one from a value. The value is changed after the statement is evaluated.

JavaScript comparison operators

Operator	Meaning
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	equal to
!=	not equal to

JavaScript comparison operators

Compare numbers:

```
let a = 10,  
    b = 20;  
  
console.log(a >= b); // false  
console.log(a == 10); // true
```

Compare strings:

```
let name1 = 'alice',  
    name2 = 'bob';  
  
let result = name1 < name2;  
console.log(result); // true  
console.log(name1 == 'alice'); // true
```

Compare a number with a value of another type:

```
console.log(10 < '20'); // true
```

```
console.log(10 == '10'); // true
```


JavaScript comparison operators

Compare an object with a non-object:

```
let apple = {
  valueOf: function () {
    return 10;
  },
};

let orange = {
  toString: function () {
    return '20';
  },
};

console.log(apple > 10); // false
console.log(orange == 20); // true
```

Compare a Boolean with another value:

```
console.log(true > 0); // true
console.log(false < 1); // true
console.log(true > false); // true
console.log(false > true); // false
console.log(true >= true); // true
console.log(true <= true); // true
console.log(false <= false); // true
console.log(false >= false); // true
```

JavaScript comparison operators

Compare null and undefined:

```
console.log(null == undefined); // true
```

Compare NaN with other values:

If either value is NaN, then the equal operator(==) returns **false**

```
console.log(NaN == 1); // false
```

```
console.log(NaN == NaN); // false
```

JavaScript comparison operators

Strict equal (===) and not strict equal (!==):

The strict equal and not strict equal operators behave like the equal and not equal operators except that they don't convert the operand before comparison

Operator	Meaning
===	strict equal
!==	not strict equal

```
console.log("10" == 10); // true  
console.log("10" === 10); // false
```

JavaScript Logical Operators

- The **NOT** operator (!) negates a boolean value.
- The **AND** operator (&&) is applied to two Boolean values and returns **true** if both values are **true**.
- The **OR** operator (||) is applied to two Boolean values and returns **true** if one of the operands is **true**.

a	b	a && b
true	true	true
true	false	false
false	true	false
false	false	false

a	b	a b
true	true	true
true	false	true
false	true	true
false	false	false

JavaScript Logical Operators

```
console.log(!undefined); // true
console.log(!null); // true
console.log(!20); //false
console.log(!0); //true
console.log(!NaN); //true
console.log(!{}); // false
console.log(!''); //true
console.log(!'OK'); //false
console.log(!false); //true
console.log(!true); //false
```

```
let eligible = false,
    required = true;

console.log(eligible && required); // false
```

```
let eligible = true,
    required = false;

console.log(eligible || required); // true
```

Primitive vs reference values

JavaScript has two different types of values:

- Primitive values (null, undefined, boolean, number, string, symbol, and BigInt)
- Reference values that refer to objects.

JavaScript allocates a fixed amount of memory space to the static data and store it on the stack

JavaScript stores objects (and functions) on the heap → The JavaScript engine doesn't allocate a fixed amount of memory for these objects. Instead, it'll allocate more space as needed.

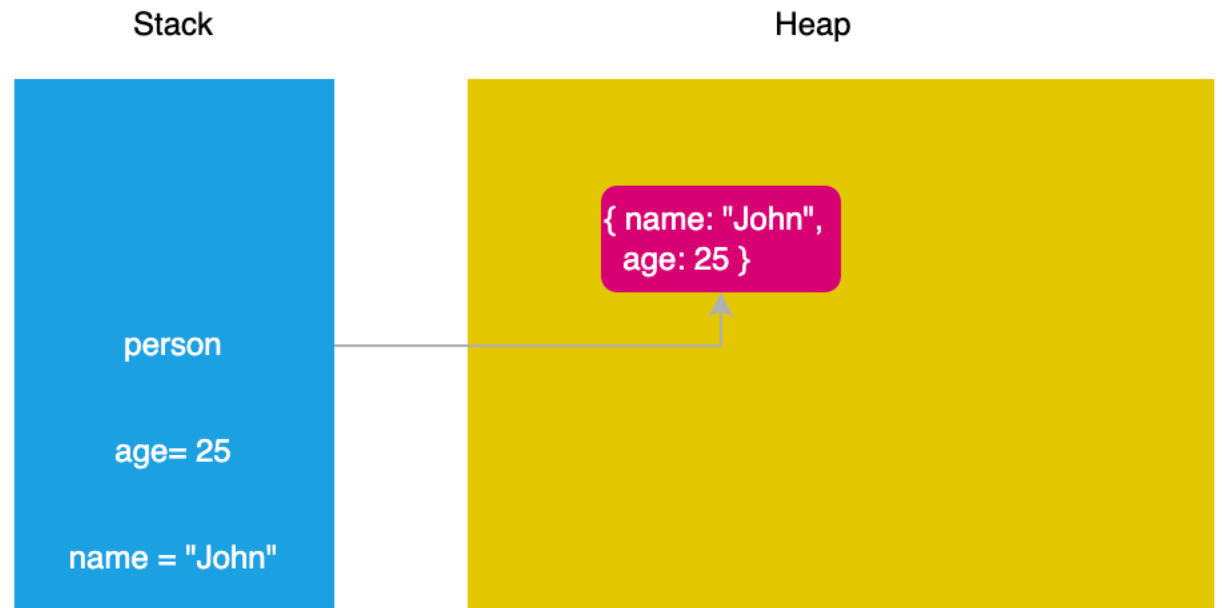
You can add, change, or delete properties to a reference value, whereas you cannot do it with a primitive value.

Copying a primitive value from one variable to another creates a separate value copy.

Copying a reference from one variable to another creates a reference so that two variables refer to the same object.

Primitive vs reference values

```
let name = 'John';  
let age = 25;  
  
let person = {  
  name: 'John',  
  age: 25,  
};
```



Primitive vs reference values

```
let person = {  
  name: 'John',  
  age: 25,  
};  
  
let member = person;
```

