(I) Syntax

1. Khai báo biến

- **Var:**
  - Là biến toàn cục
  - Dù có khai báo ở đâu thì cũng được đem lên đầu scope trước khi code được thực hiện.
  - ví dụ với đoạn code này

```
console.log (greeting);
var greeting = "say hello";
```

  - Sẽ được biên dịch là

```
var greeting;
console.log(greeting); // greeting is undefined
greeting = "say hello";
```

  - **Var** tự khởi tạo biến không được gán giá trị với giá trị là undefine

- **Let:**
  - Là biến cục bộ. ví du:

```
let times = 4;

if (times > 3) {
     let hello = "say Hello instead";
     console.log(hello); // "say Hello instead"
 }
console.log(hello); // hello is not defined
```

  - **Let** cho phép chúng ta cập nhập giá trị của biến chứ không cho phép khai báo lại biến đó.
  - **Let** không có bất cứ giá trị nào được khởi tạo

- **Const:**
  - Là biến hằng, không cho phép khai báo lại hay thay đổi giá trị của biến.

2. Kiểu dữ liệu

- **Kiểu Number :** Bao gồm tất cả số nguyên, số thực, ….
- **Kiểu String :** String là kiểu dữ liệu dùng để biểu diễn chữ, văn bản, đoạn văn bản,..
- **Kiểu Boolean** : Cho 2 giá trị true hoặc false
- **Kiểu Array :** Cho phép lưu nhiều giá trị vào 1 biến.
- **Kiểu Object :** Cho phép lưu nhiều loại kiểu dữ liệu phức tạp
- **Null :** Cho thấy là biến tồn tại nhưng không có giá trị.
- **Undefined :** Biến không tồn tại.

(II) Toán Tử

- **Phép gán:** **-** let a = 10

  - var a = 10

- Cộng, trừ, nhân(*), chia (/), chia lấy dư (%)

- let a = a + 1

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
<<=	x <<= y	x = x << y
>>=	x >>= y	x = x >> y
>>>=	x >>>= y	x = x >>> y
&=	x &= y	x = x & y
^=	x ^= y	x = x ^ y
|=	x |= y	x = x | y
**=	x **= y	x = x ** y

(III) In ra : console.log(); document.write(); alert()

(IV) Nhận dữ liệu từ người dùng:

- confirm() và prompt()
- - popup thông báo
  - popup thông báo + Yes/No
  - popup thông báo + nhập nội dung

(V) If else / switch

Syntax

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

(VI) Vòng lặp LOOP với for

- Vòng lặp for:

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

Example

```
for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}
```

- Vòng for với array:
  - #### Syntax

    ```
    for (variable in array) {
      code
    }
    ```

  - #### Example

    ```
    const numbers = [45, 4, 9, 16, 25];

    let txt = "";
    for (let x in numbers) {
      txt += numbers[x];
    }
    ```

- Vòng lặp for với object:

- Syntax

```
for (key in object) {
  // code block to be executed
}
```

- Example

```
const person = {fname:"John", lname:"Doe", age:25};

let text = "";
for (let x in person) {
  text += person[x];
}
```

- Foreach
  - Example

```
const numbers = [45, 4, 9, 16, 25];

let txt = "";
numbers.forEach(myFunction);

function myFunction(value, index, array) {
  txt += value;
}
```

- For of
  - Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

  - Example

```
const cars = ["BMW", "Volvo", "Mini"];

let text = "";
for (let x of cars) {
  text += x;
}
```

  - Example

```
let language = "JavaScript";

let text = "";
for (let x of language) {
text += x;
}
```

(VII) Vòng lặp Loop với while

- Chung:
  - Syntax

```
while (condition) {
  // code block to be executed
}
```

  - Example

```
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

- Do while:
  - Syntax

```
do {
  // code block to be executed
}
while (condition);
```

  - Example

```
do {
  text += "The number is " + i;
  i++;
}
while (i < 10);
```

(VIII) Function

- Chung
  - Example

```
function myFunction(p1, p2) {
  return p1 * p2;   // The function returns the product of p1 and p2
}
```

(IX) Object

- là một đối tượng chứa rất nhiều thuộc tính, là tất cả các kiểu dữ liệu

Example

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

- Truy cập vào các thuộc tính trong đối tượng:

Example

```
name = person.fullName();
```

  - Nếu truy cập tới hàm vào không có dấu ngoặc nó sẽ trả về kiểu định nghĩa của hàm :

Example

```
name = person.fullName;
```

(X) Chuỗi

- Khai báo

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length;
```

- Biến kí tự đặc biệt thành chuỗi:

Code	Result
\'	'
\"	"
\\	\

Code	Result
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Horizontal Tabulator
\v	Vertical Tabulator

- Định nghĩa chuỗi như 1 đối tượng:

```
let y = new String("John");
```

- Methods string
  - `slice(start, end)`

    Trả về chuỗi mới với nội dung chuỗi cũ bắt đầu từ start đến kết tại end. Nếu bỏ qua đối số end, method sẽ lấy từ end cho tới cuối chuỗi

```
let str = "Apple, Banana, Kiwi";
let part = str.slice(7, 13);
```

=> banana

  - `substr(start, length)`

Trả về chuỗi mới với nội dung chuỗi cũ bắt đầu từ start, với số ký tự sao chép là length , nếu bỏ qua tham số thứ 2, nó sẽ không lấy phần còn lại của chuỗi cũ

Example

```
let str = "Apple, Banana, Kiwi";
let part = str.substr(7, 6);
```

  - Replace()

    Thay thế từ tìm được bằng 1 từ được chỉ định

Example

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "W3Schools");
```

  - Nối Chuỗi : concat()

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2);
```

=>Hello World

  - Phương thức indexOf () trả về chỉ số của (vị trí của) lần xuất hiện đầu tiên của một văn bản được chỉ định trong một chuỗi:

Example

```
let str = "Please locate where 'locate' occurs!";
str.indexOf("locate");
```

=> 7

  - Phương thức include () trả về true nếu một chuỗi chứa một giá trị được chỉ định.

Example

```
let text = "Hello world, welcome to the universe.";
text.includes("world");
```

(XI) Number

- Cộng 1 số với 1 chuỗi = > chuỗi
  - ```
    let x = "10";
    let y = 20;
    let z = x + y;
    ```
- Nhân, chia, trừ 1 số với 1 chuỗi hoặc 2 chuỗi với nhau = > 1 số:
  - ```
    let x = "100";
    let y = "10";
    let z = x / y;
    ```
    =>10
  - ```
    let x = "100";
    let y = "10";
    let z = x * y;
    ```
    =>1000
  - ```
    let x = "100";
    let y = "10";
    let z = x - y;
    ```
    =>90
  - Các chuỗi không phải là chữ số, kết quả sẽ trả về NaN

    Example

    ```
    let x = 100 / "Apple";
    isNaN(x);
    ```
    => true

(XII) Mảng

- Khai báo mảng : let a = []

  Syntax:

  ```
  const array_name = [item1, item2, ...];
  ```
  - Hoặc
- Khai báo động

  Example

  ```
  const cars = new Array("Saab", "Volvo", "BMW");
  ```
- Thêm phần tử cho mảng vào vị trí cuối :

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi");
```

- Xóa phần tử cuối cùng của mảng:

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
```

- xóa phần tử đầu tiên

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();
```

- thêm và xóa phần tử :

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi");
```

  - Chỉ mục 2 : vị trí bắt đầu thêm 2 giá trị “lemon” và Kiwi
  - Chỉ mục 0: kể từ sau vị trí số 2, xóa
- Tạo 1 mảng mới bằng cách nối 2 mảng hiện có:

Example (Merging Two Arrays)

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];

const myChildren = myGirls.concat(myBoys);
```

- Cắt một phần của mảng tạo thành mảng mới:

```
1  const fruits = ["Banana", "Orange", "Apple", "Mango"];
2  var newfruits = fruits.slice(1,3)
3  console.log(newfruits)
```

CONSOLE ×

```
▸ (2) ["Orange", "Apple"]
```

- Đưa mảng về chuỗi

```
1  var fruits = ["Banana", "Orange", "Apple", "Mango"];
2  var tmp = fruits.toString();
3  console.log(tmp)
```

CONSOLE ×

```
Banana,Orange,Apple,Mango
```

(XIII) Tìm hiểu về async, promise.

- Đồng bộ **Sync** : hiểu đơn giản là luồng logic chạy đúng thứ tự
  -
```
console.log(1)
console.log(2)
```
```
CONSOLE ×
1
2
```
- Không đồng bộ **Async** : Logic chạy không theo thứ tự
  -
```
setTimeout(function(){
  console.log(1)
},1000)
console.log(2)
// sau khi in ra 2 xong, 1s sau mới in ra 1
```
```
CONSOLE ×
2
1
```
- **Promise** sinh ra để làm việc với không đồng bộ: (như kiểu là 1 lời hứa từ tương lai và trả về lời hứa đó có được thực thi hay không)
  - Cách hoạt động
  -
```
var promise = new Promise(function(resolve, reject){
    // logic
    // resolve : trạng thái thành công
    //  reject : trạng thái thất bại
    resolve("ok!");
  }
);
promise
        .then(function(){
          // nếu trạng thái trả về là resolve thì .then sẽ được gọi
          console.log("successfully")
        }
        .catch(function(){
          // nếu trạng thái trả về là reject thì catch sẽ được gọi
          console.log("error")
        })
        .finally(function(){
          // nếu resolver hoặc reject được gọi thì finally sẽ được gọi
          console.log("done")
        })
```

(XIV) arrow function

  -
```
var sum = (a,b) => a+b
console.log(sum(1,2))
```
```
CONSOLE ×
3
```
- Nếu sau dấu ‘ => ’ là cặp {} thì bắt buộc phải có return

```
var sum = (a,b) => {
  return a+b
}
console.log(sum(1,2))
```
```
CONSOLE ×

3
```

- Nếu muốn trả về 1 object, thêm ngoặc tròn bên ngoài object:

```
var sum = (a,b) => ({a:a , b:b })
console.log(sum(1,2))
```
```
CONSOLE ×

▸ (2) {a: 1, b: 2}
```

- Arrow function không có context (đối tượng this):
  - Với hàm bình thường

```
var users = function(username,password){
    this.username = username
    this.password = password
}
let user = new users('admin','123456')
console.log(user)
```
```
CONSOLE ×

▸ users {username: "admin", password: "12345...}
```

  - Khi sử dụng arrow func sẽ bị lỗi, vì không được dùng:

```
var users = (username,password) => ({
    this.username = username
    this.password = password
})
let user = new users('admin','123456')
console.log(user)
```
```
CONSOLE ×                                                    ...

Expected ":" but found "."                                   script.js? 2:8
```

(XV) spread operator

```
var arr1 = ['stupid','kill','you']
var arr2 = ['hacker','lord']
// nối chuỗi 2 với chuỗi 1 bằng cách dùng toán tử spread
var arr3 = [...arr2,...arr1] // ...arr đơn giản là bỏ 2 cái ngoặc [] và chỉ còn nội dung bên trong
console.log(arr3)
```
```
CONSOLE ×                                    ...   WEBSITE VIEW ×

▾ (5) ["hacker", "lord", "stupid", "kill",...]
  0: "hacker"
  1: "lord"
  2: "stupid"
  3: "kill"
  4: "you"
```

(XVI) rest parameter

- Có 2 sối số a,b đứng trước ...rest. Nên 2 giá trị đầu tiên sẽ được gán cho a,b. Các gtri còn lại tạo thành mảng và gắn cho rest

```
function parameter (a,b, ...rest){
  console.log('a =',a)
  console.log('b =',b)
  console.log(rest)
}
parameter(1,2,3,4,5)
```

```
CONSOLE ×
a = 1
b = 2
▸ (3) [3, 4, 5]
```

- Tương tự với mảng

```
function parameter ([a,b, ...rest]){
  console.log('a =',a)
  console.log('b =',b)
  console.log(rest)
}
arr = [1,2,3,4,5]
parameter(arr)
```

```
CONSOLE ×
a = 1
b = 2
▸ (3) [3, 4, 5]
```

- Tương tự với object

```
function parameter ({a,b, ...rest}){
  console.log('a =',a)
  console.log('b =',b)
  console.log(rest)
}
obj = {
  a:1,
  b:2,
  c:3,
  d:4
}
parameter(obj)
```

```
CONSOLE ×                    WEBSITE VIEW ×
a = 1
b = 2
▸ (2) {c: 3, d: 4}
```

(XVII) Bài tập

Làm ví dụ sau:
const a = {name: “Web training 2”, age: 1, point: 25}
yêu cầu: in ra đối tượng b = {name: “Web training 2”, point: 25}

```
a = {
  name : 'Web trainning team 2',
  age  : 1,
  point: 25
}
function obj({name,age,point}){
  return {
    name  : name,
    point :point
  }
}
b = obj(a)
console.log('b =',b)
```

CONSOLE ×

```
b = ▼ (2) {name: "Web trainning team 2", point...}
      name: "Web trainning team 2"
      point: 25
```