1. Cú pháp và Kiểu Dữ liệu

* Biến (Variables):

- var: Phạm vi của var là phạm vi hàm (function scope). Nó có thể được khai báo lại và có thể được gán lại giá trị mới.

Khuyến cáo: Hạn chế sử dụng vì dễ gây lỗi do hoisting và phạm vi không rõ ràng.

Cú Pháp: var + name variabl   
 Ví dụ:

var Long\_Dep\_Trai;

- let: Phạm vi của let là phạm vi khối (block scope), nghĩa là nó chỉ tồn tại bên trong một cặp dấu {} và có thể thay đổi giá trị.

- Khuyến cáo: Sử dụng khi bạn muốn thay đổi giá trị của biến trong phạm vi khối.

Cú Pháp: let + name variabl   
 Ví dụ:

let  Long\_Dep\_Trai;

Ví dụ: về let có thể phần tử bên trong

if(true){

let a = 'Long\_Dep\_Trai' // a được khai báo và gán cho giá trị đầu tiên

    a = 'Long\_Khong\_dep\_trai' // a gán cho một giá trị khác

    console.log(a)  // in ra Long\_Khong\_dep\_trai

}

- **const**: Tương tự như let, nhưng không thể thay đổi giá trị sau khi đã gán. Tuy nhiên, đối với đối tượng (object) hoặc mảng (array), các thuộc tính hoặc phần tử bên trong vẫn có thể thay đổi.

Cú Pháp: let + name variabl   
 Ví dụ:

const Long\_Dep\_Trai;

Ví dụ: về const không thay đổi giá trị sau khi gán

if(true){

    const b = 'Long\_Khong\_Dep\_Trai'

    b = 'rat\_dep\_trai'; // error Assignment to constant variable.

    console.log(b) // lỗi in ra

}

- Khuyến cáo: Sử dụng cho các giá trị không thay đổi hoặc đối tượng cần tính bất biến.

* Phạm vi Biến (Variable Scope):

- **Function scope**: Biến khai báo bên trong hàm chỉ tồn tại trong hàm đó.

Ví dụ:

function Longkute(){

    var named = 'Longkute22'; //khai bao biến trong function

    console.log(named) // in ra Longkute22

}

Longkute(); // gọi hàm

console.log(named) // lỗi phạm vi biến

- **Block scope**: Với let và const, biến chỉ tồn tại trong cặp {}. Có thể thấy nó xuất hiện trong các If,Else, For,…

Ví dụ: Block scope

if(true){

    let a = 'Long\_Dep\_Trai'

    const b = 'Long\_rat\_dep\_trai'

  }

- **Local variable**: Biến khai báo bên trong hàm hoặc khối.

- **Global variable**: Biến khai báo ngoài mọi hàm hoặc khối.

**- Tham Số (Parameter):** là một biến được sử dụng trong một hàm.

**- Đối Số (Argument):** là một giá trị được truyền từ người gọi đến hàm khi thực hiện lệnh gọi hàm.

* Kiểu Dữ liệu (Data Types):

- **Nguyên thủy (Primitive)**: Bao gồm các kiểu:

- **Number**: Đại diện cho cả số nguyên và số thập phân. Ví dụ: 10, 3.14.

- **String**: Đại diện cho chuỗi ký tự.

Ví dụ: "Hello World".

- **Boolean**: Chỉ có hai giá trị true hoặc false.

- **null**: Đại diện cho một giá trị rỗng có chủ đích.

- **undefined**: Một biến chưa được gán giá trị sẽ có kiểu undefined.

- **Symbol**: Dùng để tạo ra các giá trị độc nhất, không thể bị trùng lặp.

- **Tham chiếu (Reference)**:

- **Object**: Đối tượng chứa các cặp khóa-giá trị.

Ví dụ: { name: 'John', age: 30 }.

- **Array**: Một danh sách có thứ tự các phần tử, có thể là bất kỳ kiểu dữ liệu nào. Để tạo

Ví dụ: [1, 2, 3].

- **Function**: Một khối mã được định nghĩa và có thể được gọi lại ở bất kỳ đâu.

Ví dụ: function greet() { return 'Hello'; }.

* Toán tử (Operators):

- **Toán tử số học:**

- + (cộng):

  const a = 10

    const b = 5

    var addition =  a + b

  console.log(addition) // in ra sẽ bằng 15

- - (trừ):

const a = 10

    const b = 5

var subtraction = a-b

    console.log(subtraction)

- \* (nhân):

const a = 10

    const b = 5

var multiplication = a\*b

    console.log(multiplication) // in ra sẽ bằng 50

- / (chia):

const a = 10

    const b = 5

var division = a/b

    console.log(division) // in ra sẽ bằng 2

- % (phần dư):

   const a = 12

   const b = 5

   var percent = a%b

   console.log(percent) // in ra phần trăm là 2

- **Toán tử so sánh**:

- === (so sánh bằng chặt, xét cả kiểu dữ liệu).

- != (khác)

- !== (khác chặt)

- >, <, >=, <=.

- **Toán tử logic**:

- && (và)

- || (hoặc)

- ! (phủ định).

- **Toán tử gán**:

- **=** (gán giá trị)

- += (cộng thêm vào)

- -= (trừ đi)

- \*= (nhân)

- /= (chia).

- **Toán tử điều kiện (ternary)**:

- Cú pháp: condition ? expr1 : expr2.

- Nếu condition đúng, trả về expr1, nếu sai, trả về expr2.

Ví dụ:

let tuoi = 18;

let kiem\_Tra = tuoi >= 18 ? "Đã đủ tuổi" : "Chưa đủ tuổi"

console.log(kiem\_Tra) // in ra đã đủ tuổi

1. **Cấu trúc Điều khiển**

* **Điều kiện (Conditionals):**

- `if`, `else if`, `else`:

- Dùng để thực hiện các khối mã dựa trên điều kiện đúng hoặc sai. Không nên dùng quá 3 vòng lặp if else

- Ví dụ:

- `switch`:

- Dùng khi bạn có nhiều điều kiện cần kiểm tra giá trị của một biến.

- Ví dụ:

* **Vòng lặp (Loops):**

- `for`:

- Sử dụng khi bạn biết trước số lần lặp.

- Ví dụ:

- `while`:

- Sử dụng khi bạn chưa biết số lần lặp, chỉ dừng khi điều kiện sai.

- Ví dụ:

- `do...while`:

- Giống như while, nhưng đảm bảo khối mã được thực hiện ít nhất một lần.

- Ví dụ:

- Sử dụng `break` và `continue`:

- `break`: Dừng vòng lặp ngay lập tức.

Ví dụ:

const number = [1, 2, 3, 4, 5, 6, 7, 8, 9];

count = 0;

for (let i = 0; i < number.length; i++) {

  if (number[i] % 2 !== 0) {

    break;

  }

  count++;

}

console.log(count); // in ra 0

- `continue`: Bỏ qua các lệnh phía sau và chuyển sang vòng lặp tiếp theo.

Ví dụ:

const number = [1, 2, 3, 4, 5, 6, 7, 8, 9];

count = 0;

for (let i = 0; i < number.length; i++) {

  if (number[i] % 2 !== 0) {

    continue;

  }

  count++;

}

console.log(count);

### 3. Hàm (Functions)

#### Định nghĩa và Gọi hàm:

- Định nghĩa hàm: Hàm là một khối mã có thể gọi lại nhiều lần để thực hiện 1 nhiệm vụ nào đó. Được định nghĩa bắt buộc với từ khoá “function”, có tên.   
\* Lưu ý: Cần cẩn thận sử dụng với vì nó có thể gọi nó trước khi khai báo function

- Cú pháp: function name(parameters) { // code }.

- Ví dụ:

function Longkute() { // từ khoá funtion name (parameters)){ // code }

  console.log("Hello")

}

Longkute(); // in ra Hello

- Biểu thức hàm (function expressions):

- Khai báo hàm dưới dạng biểu thức và có thể gán vào biến. Và nó chỉ có thể sử dụng phía dưới hàm đó.

- Ví dụ:

const named = "Long\_dep\_trai";

const greet = function (*name*) { // gán biến vào hàm funtion

    return console.log("Hello " + *name*);

};

greet(named); //in ra Hello Long\_dep\_trai

- Biểu thức hàm ẩn danh(**Anonymous Function**): Là một hàm không tên, thường được sử dụng trong các tình huống tạm thời.

-Thường được sử dụng trong các tình huống callback

- Các Phương thức Mảng

- IIFE(Immediately Invoked Function Expression)

- Hàm mũi tên (arrow functions): sử dụng kí hiệu =>

- Cú pháp gọn gàng hơn với các hàm không có tên.

- Ví dụ:

const arrow\_function = *a* => *a* \* *a*;

console.log(arrow\_function(12)); // in ra 144

Hoặc

const nhan = (*a*, *b*) => {

  const ket\_Qua = *a* \* *b*;

  return ket\_Qua;

};

console.log(nhan(2, 3)); // in ra 6

**- Hàm constructor**

**### 4. Đối tượng và Mảng**

**Đối tượng (Objects): là kiểu dữ liệu dữ liệu phức tạp (**reference type) chứa nhiều kiểu dữ liệu khác nhau thông qua cặp key - value. Value có thể là kiểu dữ liệu bất kỳ: number, string, boolean, array, function, object…

Cách tạo đối tuọng cần thông qua cặp: {} bên trong sẽ có có key và value

**- Cách tạo đối tượng:**

**- Bạn có thể tạo đối tượng bằng cách sử dụng cú pháp { key: value }.**

**- Truy cập và sửa đổi thuộc tính của đối tượng:**

**- Bạn có thể truy cập thuộc tính của đối tượng bằng cách sử dụng dấu chấm (.) hoặc dấu ngoặc vuông ([]).**

**- Ví dụ:**

const person = {

    name: "Long\_kute", // key: value

    age: 24,// key: value

    greet: function(){ // key: value

        console.log("Hello" + person.name )

    }

}

console.log(person); // in ra { name: 'Long\_kute', age: 24, greet: [Function: greet] }

person.greet(); // in ra

**- Ví dụ:**

**#### Mảng (Arrays):**

**- Cách tạo và truy cập phần tử trong mảng:**

**- Mảng được tạo bằng cách sử dụng dấu ngoặc vuông ([]), và mỗi phần tử trong mảng có một chỉ mục (index), bắt đầu từ 0.**

**- Ví dụ:**

const test\_Arr = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy' ] // tạo 1 mảng

console.log(test\_Arr[0]);// in ra Hoàng

**- Các phương thức phổ biến của mảng:**

**- concat: nối 2 hay nhiều mảng với nhau.**

**Ví dụ:**

const arr1 = [1,2,3,4]  // mảng 1

const arr2 =[5,6,7,8]   // mảng 2

const merge = arr1.concat(arr2)// kết hợp 2 mảng bằng phương thức concat

console.log(merge); // in ra [1,2,3,4,5,6,7,8]

**- `push()`: Thêm phần tử vào cuối mảng. Nếu dùng push để kết hợp 2 mảng với nhau thì không phù hợp vì nó sẽ thêm tất cả vào lun mà ko tách ra**

**Ví dụ:**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa có Thuý

test\_Arr1.push('Thuý'); // sử dụng phương thức push để thêm phần tử vào cuối mảng

console.log(test\_Arr1);// in ra mảng thêm Thuý vào cuối mảng

**- `unshift()`: Thêm phần tử vào đầu mảng.**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa có Thuý

  test\_Arr1.unshift('Thuý'); // sử dụng phương thức unshift để thêm phần tử vào đầu mảng

console.log(test\_Arr1);// in ra mảng thêm Thuý vào đầu mảng

**- `pop()`: Loại bỏ phần tử cuối cùng của mảng.**

**Ví dụ:**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa mất gì

  test\_Arr1.pop(); // sử dụng phương thức pop để loại phần tử cuối cùng của mảng

console.log(test\_Arr1);// in ra mảng  ['Hoàng', 'Long', "Phát", 'Linh' ]

**- `shift()`: Loại bỏ phần tử đầu tiên của mảng.**

**Ví dụ:**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa mất gì

    test\_Arr1.shift(); // sử dụng phương thức shift để loại phần tử đầu tiên của mảng

console.log(test\_Arr1);// in ra mảng  [ 'Long', "Phát", 'Linh', 'Quy']

**- `slice()`: Trích xuất một phần của mảng mà không làm thay đổi mảng gốc. Sẽ có bắt đầu và kết thúc.  
Ví dụ:**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy', 'Siêu' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa mất gì

     // sử dụng phương thức slice trích xuất 1 phần của mảng mà không thay đổi mảng gốc

console.log(test\_Arr1.slice(2,4));// in ra mảng  [ 'Phát', 'Linh' ]

console.log(test\_Arr1.slice(2,6));// in ra mảng  [ 'Phát', 'Linh', 'Quy', 'Siêu' ]

console.log(test\_Arr1.slice());// in ra mảng  ['Hoàng', 'Long', "Phát", 'Linh', 'Quy', 'Siêu' ]

**- `splice()`: Thêm, xóa hoặc thay thế các phần tử trong mảng.**

Cú pháp: splice(start, deleteCount, item1, item2, /\* …, \*/ itemN)

**Ví dụ:**

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy', 'Siêu' ] // tạo  mảng 1

console.log(test\_Arr1 ); // in ra mang chưa mất gì

     // sử dụng phương thức splice thêm hoặc xoá các phần tử trong mảng

test\_Arr1.splice(2, 0, 'Super', 'Siuuuuu')

console.log(test\_Arr1);// in ra mảng  [ 'Hoàng', 'Long', 'Super', 'Siuuuuu', 'Phát',  'Linh', 'Quy',   'Siêu']

**- `map()`:** Được sử dụng để lặp qua từng phần tử trong mảng và trả về một mảng mới chứa các giá trị đã được biến đổi.

const test\_Arr1 = ['Hoàng', 'Long', "Phát", 'Linh', 'Quy', 'Siêu' ] // tạo  mảng 1

const map1 = test\_Arr1.map((*x*) => {

    if( *x* == "Hoàng") {

      return  *x* = "Niên" // in ra [ 'Niên', undefined, undefined, undefined, undefined, undefined ]

    } else {

    return *x* // nếu không có return x thì sẽ in ra [ 'Niên', undefined, undefined, undefined, undefined, undefined ]

    }

})

console.log(map1);// in ra [ 'Niên', 'Long', 'Phát', 'Linh', 'Quy', 'Siêu' ]

**- `filter()`:** Dùng để lọc các phần tử trong mảng theo một điều kiện nhất định và tạo ra một mảng mới chỉ chứa các phần tử thỏa mãn điều kiện đó.

**- `reduce()`: Áp dụng một hàm lên tất cả các phần tử trong mảng và trả về một giá trị duy nhất.**

**- find : lọc phần tử trongmảng nhưng sẽ trả về giá trị đầu tiên nó tìm thấy trong mảng hoặc trả về undefined nếu không tìm thấy**

**- foreach():**lặp qua các phần tử của một mảng và thực hiện một hành động nào đó đối với mỗi phần tử

**-**

**### 5. Tương tác với HTML và DOM**

**#### DOM Manipulation:**

**- Truy cập các phần tử HTML:**

**- Bạn có thể truy cập các phần tử HTML thông qua DOM (Document Object Model) bằng các phương thức như:**

**- document.getElementById('id'): Lấy phần tử theo ID.**

**- document.querySelector('selector'): Lấy phần tử dựa trên CSS selector.**

**Ví dụ:**

**const header = document.getElementById('header');**

**const button = document.querySelector('.btn');**

**- Thay đổi nội dung và thuộc tính của các phần tử:**

**- Bạn có thể thay đổi nội dung và các thuộc tính của các phần tử đã truy cập.**

**- Ví dụ:**

**header.textContent = 'New Header Text'; // Thay đổi nội dung**

**button.style.backgroundColor = 'red'; // Thay đổi thuộc tính CSS**

**#### Sự kiện (Events):**

**- Thêm sự kiện:**

**- Bạn có thể thêm các sự kiện bằng cách sử dụng addEventListener().**

**- Ví dụ:**

**button.addEventListener('click', () => {**

**alert('Button clicked!');**

**});**

**- Các loại sự kiện phổ biến:**

**- `click`: Xảy ra khi phần tử bị click.**

**- `mouseover`: Xảy ra khi di chuột qua phần tử.**

**- `keyup`: Xảy ra khi nhả phím trên bàn phím.**

**- `submit`: Xảy ra khi một form được gửi đi.**

**---**

**### 6. Lập trình Bất đồng bộ (Asynchronous Programming)**

**#### Callbacks:**

**- Hàm callback là một hàm được truyền vào hàm khác như một đối số và được thực thi sau khi hàm khác hoàn tất công việc của nó.**

**- Ví dụ:**

function fetch(*a*) {

  const profile = { name: "Long\_kute", age: 24 };

  a(profile);

}

fetch((*fetch\_data2*) => {

  console.log(*fetch\_data2*); // dùng arrow function để tạo một định nghĩa đối số, in ra kết quả

});

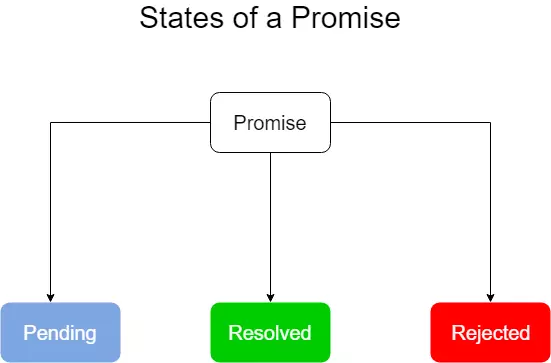
function fetch\_data(*data*) {

  console.log(*data*);

}

fetch(fetch\_data); // in ra { name: 'Long\_kute', age: 24 }

**#### Promises:**



Promise cũng giống như lời hứa trong thực tế có 3 trạng thái: chưa được giải quyết (unresolved), đã giải quyết (resolved) , hoặc bị từ chối (rejected)

**- Promises cung cấp cách xử lý mã bất đồng bộ dễ dàng hơn so với callback.**

**- Tạo Promise:**

**- Sử dụng `then()`, `catch()`, `finally()`:**

**- then(): Xử lý kết quả khi Promise thành công.**

**- catch(): Xử lý lỗi nếu Promise thất bại.**

**- finally(): Thực hiện sau khi Promise kết thúc, dù thành công hay thất bại.**

**Cú pháp: Kiểu dữ liệu + name = new Promise(**

**- Ví dụ:**

let myPromise = **new** *Promise*(function (*resolve*, *reject*) {

    const success = false;

    if(success){

       resolve("Đã đẩy dữ liệu trên local thành công");

    }else{

        reject("Lỗi trong quá trình đẩy dữ liệu trên local");

    }

})

 myPromise

 .then((*message*) => { // nếu là true sử dụng then để in ra dữ liệu thành công

    console.log(*message*);

 })

 .catch((*error*) => { // nếu là false sử dụng catch để in ra lỗi

    console.log(*error*);

 })

 .finally((*check*) => { // finally luôn luôn chạy, kể cả hoàn thành hay thất bại

    console.log("Promise thành công");

 })

**#### Async/Await:**

**- `async` và `await` là cách sử dụng Promises dễ đọc hơn.**

**- Cú pháp:**

**- Từ khóa async trước một hàm sẽ làm hàm đó trả về Promise.**

**- Từ khóa await được sử dụng để tạm dừng việc thực thi cho đến khi Promise hoàn thành.**

**-**

**- Ví dụ:**

**---**