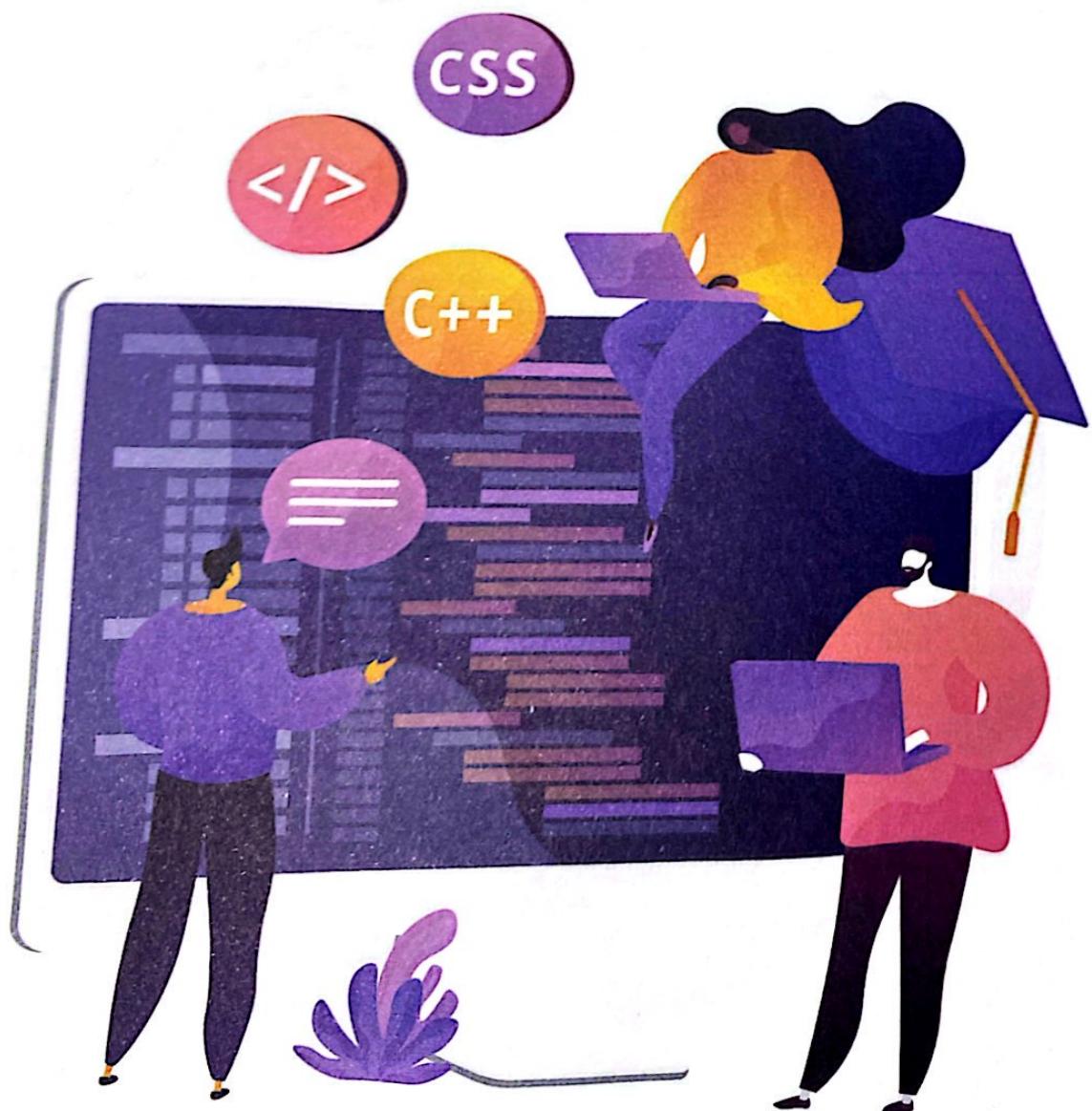


PHẦN 3:

Kiến thức, cách áp dụng,
bài tập thực hành - ngôn ngữ
JavaScript



PHẦN 3:

KIẾN THỨC, CÁCH ÁP DỤNG, BÀI TẬP THỰC HÀNH - NGÔN NGỮ JAVA SCRI

Chương 1: Tổng quan về ngôn ngữ JavaScript

I. Lý do phải học ngôn ngữ lập trình Java Script?

Có nhiều lý do để học JavaScript (JS), đặc biệt là nếu bạn quan tâm đến việc phát triển các ứng dụng web hoặc trang web động. Dưới đây là một số lý do quan trọng:

- Sử dụng rộng rãi trên web: JavaScript là ngôn ngữ lập trình phía client phổ biến nhất và được sử dụng rộng rãi trong phát triển web. Các trang web phức tạp hiện đại như Gmail, Facebook và Twitter đều sử dụng JavaScript để tạo ra các tính năng động và tương tác trên trang.
- Khả năng tương tác cao: JavaScript cho phép bạn tương tác với người dùng và thay đổi nội dung của trang web mà không cần tải lại trang. Bạn có thể sử dụng JavaScript để tạo ra các hộp thoại cảnh báo, xử lý biểu mẫu và thay đổi nội dung của trang mà không cần phải tải lại trang.
- Hỗ trợ cho nhiều framework: JavaScript được sử dụng trong nhiều framework phổ biến, bao gồm AngularJS, React và Vue.js, giúp bạn dễ dàng phát triển các ứng dụng web phức tạp.
- Tiềm năng tuyệt vời cho việc làm: Các nhà phát triển JavaScript là một trong những vị trí có nhu cầu cao nhất trong ngành công nghiệp phát triển phần mềm. Khi bạn học JavaScript, bạn có thể tìm kiếm việc làm trong các công ty phát triển web và các công ty phần mềm khác.
- Học tốt các kỹ năng lập trình cơ bản: Học JavaScript sẽ giúp bạn hiểu các khái niệm cơ bản của lập trình, bao gồm biến, điều kiện, vòng lặp và hàm. Điều này sẽ giúp bạn dễ dàng tiếp cận với các ngôn ngữ lập trình khác trong tương lai.

- › JavaScript là ngôn ngữ dễ học: JavaScript được coi là một trong những ngôn ngữ lập trình dễ học nhất. Nó có cú pháp đơn giản và cung cấp nhiều tài liệu và công cụ hỗ trợ học tập.

Vì vậy, học JavaScript sẽ giúp bạn có thể phát triển các ứng dụng web phức tạp và có tiềm năng tốt cho việc làm trong ngành công nghiệp phát triển phần mềm.

II. Lịch sử phát triển ngôn ngữ lập trình Java Script?

JavaScript (JS) được tạo ra bởi Brendan Eich của công ty Netscape vào năm 1995. Ban đầu, ngôn ngữ này được gọi là Mocha, sau đó đổi tên thành LiveScript, và cuối cùng được đổi tên thành JavaScript.

Ban đầu, JavaScript được thiết kế để làm việc trên các trình duyệt web Netscape, nhưng nhanh chóng trở thành một ngôn ngữ lập trình phía client phổ biến trên web. Ngôn ngữ này đã tiếp tục phát triển và được chuẩn hóa bởi Hiệp hội Tiêu chuẩn Hóa Quốc tế (ISO) và Tổ chức Internet (IETF).

Năm 1997, Netscape đã chuyển giao JavaScript cho tổ chức tiêu chuẩn hóa European Computer Manufacturers Association (ECMA). ECMA đã đưa ra tiêu chuẩn JavaScript đầu tiên và phát hành phiên bản chuẩn JavaScript đầu tiên vào năm 1999, được gọi là ECMAScript 1.0. Từ đó, các phiên bản ECMAScript tiếp theo được phát hành như ECMAScript 2, 3, 4 và 5.

ECMAScript 3, phát hành vào năm 1999, là phiên bản JavaScript phổ biến nhất và được sử dụng rộng rãi trên web trong nhiều năm. Tuy nhiên, vào năm 2005, Ajax (Asynchronous JavaScript and XML) trở nên phổ biến và yêu cầu nhiều tính năng mới của JavaScript. Đó là lý do tại sao ECMAScript 5 đã được phát hành vào năm 2009, cung cấp nhiều tính năng mới và được hỗ trợ rộng rãi trên các trình duyệt web hiện đại.

ECMAScript 6 (hay còn gọi là ECMAScript 2015) đã được phát hành vào năm 2015, với nhiều cập nhật quan trọng, bao gồm khả năng khai báo biến mới (let và const), mô-đun, bộ giá trị đối tượng, và nhiều tính năng khác. Các phiên bản ECMAScript tiếp theo như ECMAScript 2016, 2017, 2018 và 2019 cũng được phát hành, cung cấp nhiều tính năng mới và được sử dụng rộng rãi trên web hiện nay.

Vậy làm cách nào để cài đặt môi trường hỗ trợ học tập và tạo nên những sản phẩm riêng của bản thân.

III. Các bước để cài đặt môi trường lập trình JavaScript (JS)

- **Bước 1: Cài đặt trình duyệt web**

Trình duyệt web là công cụ tốt nhất để kiểm tra và chạy các chương trình JavaScript. Các trình duyệt web phổ biến nhất là Google Chrome, Mozilla Firefox và Microsoft Edge. Bạn có thể tải trình duyệt web miễn phí từ trang chủ của chúng.

- **Bước 2: Cài đặt trình soạn thảo mã nguồn JavaScript**

Để lập trình JavaScript, bạn cần một trình soạn thảo mã nguồn chuyên nghiệp. Các trình soạn thảo mã nguồn phổ biến nhất là Visual Studio Code, Sublime Text, Atom và Notepad++. Bạn có thể tải trình soạn thảo mã nguồn miễn phí từ trang chủ của chúng.

Sau khi cài đặt trình duyệt web và trình soạn thảo mã nguồn, bạn đã có môi trường lập trình JavaScript đầy đủ để bắt đầu học và phát triển các ứng dụng web và ứng dụng di động.

Chương 2: Kiến thức, cách áp dụng và bài tập thực hành

I. JavaScript là gì?

Trước khi bắt đầu, chúng ta sẽ điểm lại khái niệm về ngôn ngữ lập trình JavaScript: JavaScript là một trong những công nghệ chủ yếu để xây dựng các ứng dụng website. JavaScript được nhúng vào trong HTML để tăng cường trải nghiệm người dùng bằng cách thêm nhiều tương tác với ứng dụng.

Với các ứng dụng web hiện đại, vai trò của JS càng ngày càng quan trọng hơn. Toàn bộ ứng dụng có thể được tạo ra bằng JavaScript.

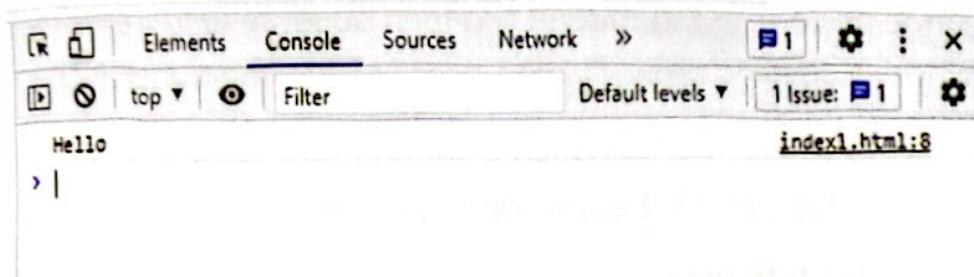
Chúng ta biết rằng, JavaScript còn có thể được sử dụng để tạo ra các ứng dụng chạy bên phía server (chạy trên máy tính thay vì trình duyệt) hay thậm chí cả các ứng dụng di động trên IOS và Android.

II. Để chạy ứng dụng JavaScript đầu tiên

Thẻ **<script>** được sử dụng để cho phép chúng ta viết code JS bên trong một trang web HTML.

Ví dụ:

```
index1.html > html
1  <html>
2  <head>
3  |  <title>First JS application</title>
4  |</head>
5  <body>
6  |
7  |  <script>
8  |  |  console.log("Hello")
9  |  </script>
10 </body>
11 </html>
```



Để xem được kết quả, mở cửa sổ console bên trong công cụ phát triển của trình duyệt.

Ngoài ra, JavaScript còn có thể được viết thành một module riêng biệt và import vào bên trong ứng dụng HTML. Đầu tiên, ta tạo ra một file JavaScript trong thư mục ứng dụng:

```
main.js
1  console.log("Hello")
2
```

Sau đó, để liên kết vào trong ứng dụng, ta sử dụng cú pháp với thẻ script như sau:

```
index1.html > html
1  <html>
2  <head>
3  <title>First JS application</title>
4  </head>
5  <body>
6  ...
7  <script src=".//main.js"></script>
8  </body>
9  </html>
```

Ta có thể thấy kết quả nhận được vẫn giống với ví dụ ở trên.

Trong một ứng dụng HTML, ta có thể có nhiều thẻ **<script>** khác nhau, cũng như liên kết nhiều file .js khác nhau. Nhờ đó, ta có thể chia nhỏ ứng dụng thành nhiều file JavaScript để dễ bảo trì sau này.

Thứ tự thực thi code thông thường của trình duyệt sẽ là từ trên xuống dưới. Với những file **JS** khác nhau, file nào được liên kết vào trước sẽ chạy trước.

Lưu ý: Chúng ta nên có thói quen chia tách ứng dụng thành các file **JS** nhỏ hơn. Ứng dụng sẽ dễ đọc, dễ bảo trì hơn sau này.

III. Biến và Khai báo biến

Biến (variable) là một trong những khái niệm cơ bản của bất cứ ngôn ngữ lập trình nào. Biến là đại diện cho một vùng nhớ trong ứng dụng. Biến được sử dụng để lưu trữ một dữ liệu, và có thể truy xuất tới dữ liệu đó qua tên của biến.

Để khai báo một biến, trong JavaScript hiện tại có 3 cách với các từ khoá như sau:

main1.js > ...

```
1 let x  
2 var y  
3 const z  
4 |
```

Những cách trên có sự khác nhau về ý nghĩa. Ta sẽ nói về chúng ở những phần tiếp theo.

Thế nào là Comment?

Trong JS, để comment một dòng lệnh, người ta có thể sử dụng cặp ký tự `//` cho từng dòng hoặc `/* */` cho nhiều dòng. Những đoạn nằm trong comment sẽ bị bỏ qua, không được chạy trên trình duyệt.

IV. Phép gán (Assignment) và hằng số

```
main1.js > ...
1 // Comment line
2
3 let x // This is a variable
4
5 /*
6 This is a multile comments.
7 Use it if you want to explain things
8 */
9 const gravity
```

```
main1.js > ...
1 let x = 10
2 console.log(x) // 10
3 x = 11
4 console.log(x) // 11
5
6 var y = 0;
7 y = 1
8 console.log(y) // 1
9
10 const GRAVITY = 9.8
11 console.log(GRAVITY) // 9.8
12 GRAVITY = 10 // Error: Assignment to constant variable
```

Gán là một thao tác cơ bản để thay đổi giá trị mà biến nắm giữ. Trong JS, phép gán được thể hiện với dấu `=`, sau đó là giá trị ta muốn gán. Phép gán cũng có thể được sử dụng trực tiếp khi biến được khai báo.

Các biến được khai báo với từ khoá `const` có một điểm khác biệt với hai từ khoá `let` và `var`: Các biến khai báo với `const` được gọi là các hằng số. Các hằng số là các giá trị không đổi trong suốt quá trình chạy của ứng dụng. Vì vậy, các hằng số này không cho phép giá trị được gán lại sau khi khởi tạo.

Trong JS, đặt tên các biến cần phải tuân thủ những quy tắc sau:

- Chữ cái đầu tiên trong tên biến buộc phải là ký tự trong bảng chữ cái

hoặc dấu gạch dưới (_) hoặc dấu đô la (\$). “**1a**” là một tên biến không hợp lệ.

- › Tên biến có thể sử dụng các chữ cái, chữ số hoặc dấu gạch dưới (_) và dấu đô la (\$).
- › Các biến có phân biệt viết hoa và viết thường (case-sensitive). ‘**a**’ và ‘**A**’ là 2 biến khác nhau.
- › Không được sử dụng các từ khoá của JS để đặt tên. Danh sách các từ khoá có thể xem tại đây:

(https://www.w3schools.com/js/js_reserved.asp)

Ngoài các quy tắc bắt buộc phải tuân theo trên, chúng ta nên tuân theo quy ước đặt tên trong JavaScript.

Với JS, quy ước đặt tên biến thường được sử dụng là dạng “**camelCase**”.

```
// bad
var firstname = 'Robin';
// bad
var first_name = 'Robin';
// bad
var FIRSTNAME = 'Robin';
// bad
var FIRST_NAME = 'Robin';
// good
var firstName = 'Robin';
```

Sau khi đã nắm được các quy ước về đặt tên biến trong JS, chúng ta sẽ tìm hiểu sâu hơn về **let & var** và thế nào là Hoisting.

Ta thấy rằng, ‘**let**’ được sử dụng thường xuyên hơn ngày nay để khởi tạo các biến có thể thay đổi giá trị. ‘**var**’ cũng là một từ khoá được sử dụng chung mục đích đó. Tuy nhiên nó có những điểm khác nhau nhất định. Một trong những đặc tính đó là **Hoisting**.

Như đã biết, code của chúng ta được thực thi từ trên xuống dưới.

Xét ví dụ sau:

```
main1.js > ...
1   console.log(x)
2   let x = 1
3
4
5
```

Với đoạn code này, ta sẽ nhận được lỗi. Vì chúng ta đã sử dụng biến 'x' trước khi nó được khởi tạo.

Mặc dù chúng ta nhận lại undefined trên màn hình console. Tuy nhiên, ứng dụng không hề gặp lỗi. Đây chính là tính năng hoisting trong JavaScript. Hoisting là hành vi JS tự động đẩy các khai báo biến lên trên đầu ứng dụng, khiến cho việc truy cập vào các biến sau đó là hợp lệ. Đây là cách thực tế đoạn code trên được thực thi.

```
main1.js > ...
1   console.log(x) // undefined
2   var x = 1
3
4
```

Các biến được khai báo với từ khoá **var** sẽ có tính chất này. Ngược lại, các biến khai báo với từ khoá **let** sẽ không có tính chất này.

Note: Vì tính chất khó đoán hơn của '**var**', trong hầu hết các ứng dụng ngày nay, người ta sử dụng '**let**' thay vì sử dụng '**var**'. Các hằng số khai báo với '**const**' cũng không có tính chất hoisting này.

Liệu ngôn ngữ JavaScript có các kiểu dữ liệu cơ bản nào ?

V. JavaScript có các kiểu dữ liệu cơ bản (nguyên thuỷ) sau

- › Number: Kiểu số. Sử dụng để thực hiện các tính toán về toán học.
- › String: Kiểu chuỗi ký tự. Sử dụng trong các xử lý liên quan tới văn bản.
- › Boolean: Kiểu dữ liệu nhị phân. Đại diện cho đúng hoặc sai.

- › Null: Kiểu dữ liệu rỗng.
- › Undefined: Kiểu dữ liệu không xác định.

Với một biến trong JavaScript, chúng không có kiểu dữ liệu cố định. Thay vào đó, các biến JavaScript sẽ có kiểu dữ liệu dựa vào giá trị mà chúng đang mang. Để biết được kiểu dữ liệu hiện tại của một biến, ta hãy sử dụng từ khoá ‘**typeof**’. Với các kiểu dữ liệu khác nhau, chúng ta sẽ có nhiều các phép tính toán khác nhau.

```
main1.js > ...
1  let x = 10
2  console.log(typeof x) // number
3
4  x = "Hello"
5  console.log(typeof x) // string
6
7  x = true
8  console.log(typeof x) // boolean
9
10 x = null
11 console.log(typeof x) // null
12
13 x = undefined
14 console.log(typeof x) // undefined.....
```

VI. Các toán tử có trong ngôn ngữ lập trình JavaScript

- › Toán tử toán học: Cộng (+), Trừ (-), Nhân (*), Chia (/), Mũ (**), Chia lấy dư (%), Tăng / Giảm 1 đơn vị (++ / --).
- › Toán tử gán : được sử dụng để gán nhanh một biến với giá trị mới.

=	+=	-=	*=	/=	%=	**=
x = y	x += y	x -= y	x *= y	x /= y	x %= y	x **= y

Ví dụ:

main1.js > ...

```
1 let x = 0  
2 x += 2  
3 console.log(x) // 2  
4  
5
```

- Toán tử nối chuỗi: Hai hoặc nhiều giá trị string có thể được nối với nhau thông qua toán tử `+`.

main1.js

```
1 x = "Hello"  
2 y = "World"  
3 console.log(x + " " + "World")  
4  
5
```

- Toán tử so sánh: Sử dụng để so sánh các dữ liệu với nhau.
 - Bằng : ==,
 - Bằng giá trị và kiểu dữ liệu : ===,
 - Không bằng : !=,
 - Không bằng giá trị và cùng kiểu dữ liệu : !==,
 - Lớn hơn & lớn hơn hoặc bằng : > & >=,
 - Bé hơn & bé hơn hoặc bằng : < & <=.

Ví dụ:

main1.js > ...

```
1 let x = 9  
2 let y = 10  
3  
4 console.log(y > x) // true  
5  
6  
7
```

- Toán tử logic: Sử dụng để tạo ra các kết quả logic giữa 2 giá trị.

&&		!
Và	Hoặc	Phủ định

main1.js > ...

```
1 const x = true
2 const y = false
3
4 console.log(x && y) // false
5
```



BÀI TẬP VẬN DỤNG

I. Bài tập

Bài thực hành 1: Tạo một trang hiển thị thông báo:

“Tôi là bậc thầy múa Florentino”

Bài thực hành 2: Cùng với bài trên, chúng ta hãy trích xuất nội dung code của bạn vào một file ngoài “main.js” và nằm trong cùng một thư mục.

Bài thực hành 3: Giá trị cuối cùng của tất cả các biến a, b, c và d sau đoạn code dưới đây là bao nhiêu?

```
let a = 1, b = 1;
```

```
let c = ++a; : ?
```

```
let d = b++; : ?
```

Phản tiếp theo, chúng ta sẽ tìm hiểu về các câu lệnh liên quan đến **Flow** (luồng) của JavaScript. Trong JavaScript, code sẽ chạy theo flow chủ yếu là từ trên xuống dưới. Ta có những câu lệnh để có thể viết được các đoạn code chạy theo một logic nhất định.

Đầu tiên là Khối lệnh (Block of code).

Đây là một tập hợp các câu lệnh sẽ được thực thi cùng nhau. Trong JavaScript, một khối lệnh được hiểu là các dòng code nằm trong cặp ngoặc nhọn {}.

```
main.js > ...
1   console.log("Hello, world!")
2   {
3       let x = 1;
4       console.log(x)
5   }
```

Mục đích của khối lệnh là để chia nhỏ ứng dụng thành nhiều phần có phạm vi (Scope) nhỏ hơn. Các biến nằm trong phạm vi khác nhau sẽ có giá trị khác nhau. Từ đó ta có khái niệm “**phạm vi của biến**”. Phạm vi



của biến nói về các vùng mà giá trị của biến khả dụng. Xét ví dụ sau:

```
main.js > ...
1 let x = 1
2 {
3     let y = 2
4     console.log(y) // 2
5     console.log(x) // 1
6 }
7 console.log(y) // Error
8 console.log(x) // 1
9
```

Với ví dụ trên, do biến **y** được khởi tạo bên trong khối lệnh con, nên **y** chỉ được biết tới bên trong khối lệnh đó. Ta không thể truy cập vào biến **y** ở ngoài phạm vi của nó.

Tuy nhiên, với biến **x**, nó được khởi tạo ở bên ngoài khối lệnh. **x** sẽ có phạm vi bên ngoài và bên trong tất cả những phạm vi con. Trong JS, người ta gọi đây là **block scope**.

Note: Các biến được khai báo với từ khoá **let** và **const** sẽ có phạm vi trong khối lệnh (**block scope**). Với các biến được khai báo với từ khoá **var**, chúng sẽ có **function scope**.

Tiếp theo chúng ta sẽ bước tiếp đến khái niệm mới về câu lệnh điều kiện (**Condition**).

Câu lệnh điều kiện giúp chúng ta có thể chạy một khối lệnh tương ứng khi một điều kiện được thỏa mãn hoặc không thỏa mãn. Nó tương đương với mệnh đề “**nếu ... thì ..., không thì ...**”. Trong JavaScript, có một vài cách để viết câu điều kiện dưới đây:

- › **IF:** Đây là cách cơ bản nhất để tạo ra một câu điều kiện với JS. Cú pháp của if như sau:

```
if (<condition>
{
    /* block of code */
}
```

```
main.js > ...
1 let x = 2;
2
3 if (x % 2 === 0) {
4     console.log("x is even");
5 }
6
7 if (true) {
8     console.log("true")
9 }
10
11 if ("") {
12     console.log("You will not see this line");
13 }
```

IF nhận vào một giá trị. Giá trị này sẽ được xác định tính đúng sai bởi JS. Nếu giá trị đó là đúng, khối lệnh bên trong **IF** sẽ được thực thi. Các câu lệnh bên dưới của **IF** sẽ tiếp tục được thực thi ngay cả khi điều kiện của **IF** không thỏa mãn.

Ví dụ: Các giá trị truyền vào trong câu điều kiện không nhất thiết phải có kiểu dữ liệu là boolean. Chúng có thể là các kiểu dữ liệu khác. Tính đúng sai (Truthy, Falsy) của chúng sẽ được JavaScript xác định.

- **ELSE:**

else trong JavaScript giúp chúng ta khai báo khối lệnh sẽ được thực thi khi điều kiện của **if** nhận giá trị sai.

else buộc phải dùng với **if**. Nó không thể tự đứng một mình.

```
main.js
1 if (<condition>){
2     console.log("condition is true")
3 } else {
4     console.log("condition is false")
5 }
```

- **ELSE IF:**

Chúng ta có thể sử dụng tiếp một mệnh đề **if** sau mệnh đề **else** trước đó. Giúp chúng ta có thể tạo ra một lệnh điều kiện tiếp theo nếu điều kiện trước đó bị sai.

```
main.js
1 if (condition1) {
2   // block of code to be executed if condition1 is true
3 } else if (condition2) {
4   // block of code to be executed if the condition1 is false and condition2 is true
5 } else {
6   // block of code to be executed if the condition1 is false and condition2 is false
7 }
8
```

➤ SWITCH - CASE:

‘switch’ là một cách để rút gọn câu lệnh **if - else if** trong một số trường hợp. ‘switch’ cho ta truyền vào một giá trị đầu vào. Ta có thể định nghĩa ra các hành động khác nhau khi giá trị bằng với một giá trị được định sẵn.

Ví dụ:

```
main.js > ...
1 let day;
2 switch (new Date().getDay()) {
3   case 0:
4     day = "Sunday";
5     break;
6   case 1:
7     day = "Monday";
8     break;
9   case 2:
10    day = "Tuesday";
11    break;
12   case 3:
13    day = "Wednesday";
14    break;
15   case 4:
16    day = "Thursday";
17    break;
18   case 5:
19    day = "Friday";
20    break;
21   case 6:
22    day = "Saturday";
23 }
24 console.log(day) // Friday
```

Sau mỗi một **case**, chúng ta cần sử dụng **break** để kết thúc quá trình xử lý của **case** đó.

Ta có thể khai báo thêm trường hợp **default**. Nếu tất cả các **case** đều không đúng thì **default** sẽ được thực thi.

```
main.js > ...
1 let day;
2 switch (new Date().getDay()) {
3     case 0:
4         day = "Sunday";
5         break;
6     case 6:
7         day = "Saturday";
8         break;
9     default:
10        day = "Not a weekend"
11    }
12 console.log(day) // Not a weekend
--
```

Ngoài ra, chúng ta cũng có thể nhóm các case có chung một logic xử lý với cách viết như sau:

```
main.js > ...
1 let day;
2 switch (new Date().getDay()) {
3     case 0:
4     case 6:
5         day = "Weekend";
6         break;
7     case 1:
8     case 2:
9         day = "Start of week days";
10        break;
11    case 3:
12    case 4:
13    case 5:
14        day = "End of week days";
15        break;
16    }
17 console.log(day) // Not a weekend
--
```

› Vòng lặp:

Sau khi hiểu được cách sử dụng và khái quát về câu điều kiện. Một khái niệm luôn xuất hiện và được sử dụng thường xuyên được nhắc đến tiếp theo chính là “**Vòng lặp**” (Loop).

Vòng lặp cho phép chúng ta lặp lại các tác vụ giống nhau nhiều lần. Trong JavaScript, chúng ta có thể sử dụng `for` hoặc `while` để tạo ra các vòng lặp.

Đối với “**for**”, chúng ta có cú pháp như sau:

```
for (<statement_1>; <statement_2>; <statement_3>) {  
    /* block of code */  
}
```

Trong đó:

- › statement_1: Câu lệnh được thực hiện trước khi vòng lặp diễn ra.
- › statement_2: Điều kiện để vòng lặp kết thúc.
- › statement_3: Câu lệnh được chạy sau mỗi một vòng lặp.

Ví dụ:

main.js > ...

```
1 let sum = 0  
2 for (let i = 0; i < 100; i++) {  
3     sum += i  
4 }  
5 console.log(sum)
```

- › statement_1 thường được sử dụng để khởi tạo các giá trị phục vụ cho vòng lặp. Ở **ví dụ** trên, ta khởi tạo giá trị biến `i = 0` trước khi vòng lặp được chạy.
- › statement_2 được sử dụng để xác định khi nào vòng lặp kết thúc. Trong ví dụ trên, chúng ta sẽ kết thúc vòng lặp khi $i == 100$.
- › statement_3 được gọi sau mỗi lần vòng lặp được chạy. Trong **ví dụ** trên, ta tăng giá trị của biến `i` lên một đơn vị sau mỗi vòng lặp.

Cả 3 câu lệnh trên đều “**không bắt buộc**” phải tồn tại trong câu lệnh ‘**for**’. Chúng ta có thể bỏ qua chúng với chỉ dấu `;`.

main.js > ...

```
1 let i = 0
2 for (; i < 99; i++) {
3     console.log(i)
4 }
5
6 for (let i = 0; i < 99;) {
7     console.log(i)
8     i++;
9 }
10
11 for (;;) {
12     console.log('infinite loop')
13 }
```

Đối với vòng lặp “**while**” gần tương tự với vòng lặp **for**. Điểm khác biệt là **while** chỉ cho phép một câu lệnh để xác định khi nào vòng lặp sẽ kết thúc. Nó tương đương với vòng lặp **for** khi chỉ có statement_2.

main.js > ...

```
1 let sum = 0;
2 let i = 0;
3 while (i < 99) {
4     sum += i;
5     i++;
6 }
7 console.log(sum);
```

Bên cạnh đó, “**break**” và “**continue**” cũng là hai khái niệm mà chúng ta cũng phải nhắc đến.

Vòng lặp thường sẽ chạy cho đến khi điều kiện của nó không được thỏa mãn. Tuy nhiên, nếu muốn kết thúc sớm vòng lặp, chúng ta có thể sử dụng từ khoá “**break**”. Vòng lặp sẽ ngay lập tức dừng lại ở ví trí “**break**”. Các câu lệnh còn lại của vòng lặp hiện tại cũng sẽ không được thực thi.

main.js > ...

```
1  for (let i = 0; i < 10; i++) {  
2      if (i === 3) {  
3          break;  
4          console.log('this will not run')  
5      }  
6      console.log(i); // 0 1 2  
7  }
```

Khác với “**break**”, “**continue**” cho phép chúng ta nhảy qua một vòng lặp hiện tại và thực hiện tiếp vòng lặp tiếp theo. Các câu lệnh còn lại sau **continue** cũng sẽ không được thực thi.

main.js > ...

```
1  for (let i = 0; i < 10; i++) {  
2      if (i % 2 !== 0) {  
3          continue;  
4          console.log('this will not run')  
5      }  
6      console.log(i); // 0 2 4 6 8  
7  }
```

Để hình dung rõ hơn về vòng lặp, chúng ta có thể lấy các ví dụ thực tế có sử dụng vòng lặp như: Công việc đánh số nhà, Trái đất quay quanh mặt trời, Loop in Youtube,...

Bên cạnh đó, bạn có thể luyện tập các logic điều khiển trong lập trình, chúng ta có thể truy cập và chơi thử game giải đố ở link sau:

[“https://blockly.games/maze”](https://blockly.games/maze)

Một thử thách nhỏ cho các bạn như sau:

Bài tập thực hành 4: Yêu cầu người dùng nhập một số ‘n’, sau đó:

- › In ra số chẵn, lẻ < n.
- › Tìm ra số chia hết cho 7 lớn nhất nhỏ hơn n.

- › Tính tổng các số đếm từ 1 đến n: $1+2+3+\dots+n$.
- › Kiểm tra n có phải là số nguyên tố không.
- › In ra hình vuông dấu * với cạnh là n (vuông đặc và rõng).
- › In ra tam giác cân với cạnh đáy có độ dài là n (các cạnh là dấu *) .

Bây giờ, chúng ta hãy cùng ngẫm lại các kiến thức đã học trước đó, thả lòng cơ thể để thư giãn, và sẵn sàng để tiếp tục cuộc hành trình với những kiến thức hay hơn và thú vị hơn trong ngôn ngữ lập trình JavaScript nào !!!

Kiến thức tiếp theo mà mỗi lập trình viên đều cần nắm vững đó chính là các kiểu dữ liệu có trong ngôn ngữ lập trình JS.

Array

Đầu tiên, chúng ta sẽ đề cập đến kiểu dữ liệu “**Array**” (mảng): Là một cấu trúc dữ liệu trong JavaScript. Array có thể chứa được nhiều kiểu dữ liệu khác nhau trong nó.

Giả sử chúng ta có 3 biến như sau:

```
main.js > ...
1 let car1 = "Saab";
2 let car2 = "Volvo";
3 let car3 = "BMW";
^
```

Chúng ta cần phải làm thế nào để có thể duyệt qua 3 biến này? Hoặc sẽ như thế nào nếu chúng ta có tới 300 biến thay vì 3 biến như trên?

Giải pháp là sử dụng array. Array có thể chứa nhiều hơn một giá trị. Và chúng ta có thể truy cập vào giá trị của chúng thông qua số thứ tự (index). Để khởi tạo một array, chúng ta sử dụng cú pháp sau:

```
main.js > ...
1 const cars = ["Saab", "Volvo", "BMW"]
2
```



Các phần tử bên trong một array được phân cách nhau bởi dấu phẩy (,). Sau khi đã khởi tạo array, chúng ta có thể truy cập vào các phần tử trong array dựa vào số thứ tự của chúng, được bắt đầu đếm từ 0 :

main.js

```
1 console.log(cars[1]) // "Volvo"  
2
```

Hoặc thay đổi phần tử bên trong array với phép gán:

main.js

```
1 cars[0] = "Vinfast"  
2 console.log(cars) // ["Vinfast", "Volvo", "BMW"]  
3
```

Trong JavaScript, các phần tử trong một array không nhất thiết phải cùng một kiểu dữ liệu. Chúng có thể có đa dạng kiểu dữ liệu.

main.js > ...

```
1 const arr = [1, false, "hello", null, []]  
2
```

Vậy nếu như mỗi phần tử trong array lại là chứa một array khác, ta lại có thêm một khái niệm mới đó chính là:

Mảng đa chiều

Ví dụ:

main.js > ...

```
1 const boards = [  
2   [".", "X", "."],  
3   ["X", "0", "0"],  
4   [".", "X", "0"]  
5 ]
```

Lúc này, mỗi phần tử của board là một mảng nhỏ hơn. Ta có thể truy cập vào một ô trên bàn cờ như sau:

main.js

```
1 console.log(boards[1][1]) // 0  
2
```

Lặp qua mảng

Ngoài ra, để lấy được chiều dài của mảng, chúng ta sử dụng thuộc tính **length** của chúng. Phần tử đầu tiên trong mảng sẽ ở vị trí 0, phần tử cuối cùng trong mảng sẽ ở vị trí **length - 1**.

Sau khi nắm được khái niệm về mảng, sử dụng những kiến thức về vòng lặp để triển khai dòng code lặp qua mảng như sau:

main.js > ...

```
1 const arr = [1, 2, 3, 4, 5, 6, 7, 8]  
2 for (let i = 0; i < arr.length; i++) {  
3   console.log(arr[i])  
4 }  
5  
6 let j = 0;  
7 while (j < arr.length) {  
8   console.log(arr[j])  
9   j++;  
10 }
```

Các phương thức thông dụng của mảng:

- **push()** thêm một phần tử vào phía sau của mảng.

main.js > ...

```
1 const arr = [1, 2, 3, 4]  
2 arr.push(5)  
3 console.log(arr) // [1, 2, 3, 4, 5]  
4
```

- **pop()** xoá phần tử cuối cùng của mảng.



main.js > ...

```
1 const arr = [1, 2, 3, 4]
2 arr.pop()
3 console.log(arr) // [1, 2, 3]
```

- › `shift()` xoá phần tử ở đầu của mảng.

main.js > ...

```
1 const arr = [1, 2, 3, 4]
2 arr.shift()
3 console.log(arr) // [2, 3, 4]
```

- › `unshift()` thêm phần tử vào đầu của mảng.

main.js > ...

```
1 const arr = [1, 2, 3, 4]
2 arr.unshift(0)
3 console.log(arr) // [0, 1, 2, 3, 4]
4
```

- › `concat()` nối nhiều mảng với nhau:

main.js > ...

```
1 const arr1 = [1, 2, 3]
2 const arr2 = [4, 5, 6]
3 const newArr = arr1.concat(arr2)
4 console.log(newArr) // [1, 2, 3, 4, 5, 6]
```

- › `sort()`: sắp xếp các phần tử trong mảng.

main.js > ...

```
1 const arr = [2, 3, 1]
2 arr.sort()
3 console.log(arr) // [1, 2, 3]
```

- › `splice()` xoá một phần tử ở vị trí bất kỳ trong mảng, đồng thời có thể thêm vào các phần tử khác ở vị trí đó.

main.js > ...

```
1 const arr = [1, 2, 3, 4, 5, 6]
2 arr.splice(1, 2, 7)
3 console.log(arr) // [1, 7, 4, 5, 6]
```

- `slice()` lấy mảng con trong mảng, dựa vào vị trí bắt đầu và vị trí kết thúc:

```
main.js > ...
1 const arr = [1, 2, 3, 4, 5, 6]
2 console.log(arr.slice(2, 4)) // [3, 4]
3
```

Bài tập thực hành 5: Dựa vào kiến thức đã học về Array, bạn hãy giải quyết các yêu cầu sau:

- Viết một hàm sum nhận vào một mảng các số nguyên và trả về tổng của chúng.
- Viết một hàm findMax nhận vào một mảng các số nguyên và trả về giá trị lớn nhất trong mảng đó.
- Viết một hàm removeDuplicates nhận vào một mảng các chuỗi và trả về một mảng mới loại bỏ các phần tử trùng lặp.
- Viết một hàm reverseArray nhận vào một mảng và trả về một mảng mới với các phần tử được đảo ngược vị trí.

Object

Tương tự với array, object là một dạng dữ liệu chứa những kiểu dữ liệu khác bên trong nó. Tuy nhiên, thay vì sử dụng số thứ tự như array, object sử dụng “key” để truy cập tới các giá trị bên trong của nó.

Trong thực tế, một chiếc xe ô tô là một object. Chiếc xe sẽ có nhiều “thuộc tính” khác nhau: Màu sắc, xuất xứ, thương hiệu,... và sẽ có nhiều “hành động” khác nhau: Nổ máy, di chuyển, phanh, bấm còi,...

Object có thể được sử dụng để lưu trữ thông tin về chiếc ô tô. Cú pháp để khởi tạo object như sau:

```
main.js > ...
1  const car = {
2      brand: "Toyota",
3      year: 2022,
4      color: "red"
5 }
```

Trong object, mỗi cặp giá trị brand: "Toyota" hoặc year: 2022 được gọi là 'key - value'.

Key và **Value** được phân tách bởi dấu hai chấm `:`. Một object có thể có nhiều cặp 'key - value'. Tuy nhiên, chúng lại không thể có 2 key trùng nhau. Hay nói cách khác, 'key' phải là duy nhất. Các cặp 'key - value' được phân tách nhau bởi dấu phẩy `,`. Tương tự như array, value bên trong object không nhất thiết phải cùng kiểu dữ liệu với nhau.

Để truy cập vào trong một key của object. Ta sử dụng dấu chấm, **ví dụ**:

```
main.js > ...
1  const car = {
2      brand: "Toyota",
3      year: 2022,
4      color: "red" }
5  console.log(car.brand) // "Toyota"
6  car.year = 2020
7  console.log(car) // {brand: "Toyota", year: 2020, color: "red"}
```

Object lồng nhau

Tương tự như Array, nếu một value bên trong object có thể chứa một object khác thì ta gọi đó là "**Object lồng nhau**". Tạo ra object lồng nhau như sau:

```
main.js > ...
1  const student = {
2      name: "Alice",
3      address: {
4          street: "Nguyen Chi Thanh",
5          streetNo: 70
6      }
7 }
8  console.log(student.address.streetNo) // 70
```

Tương tự bên trên, chúng ta cũng có thể thay đổi giá trị bên trong object lồng nhau thông qua phép gán.

Các thao tác với Object mà chúng ta cần biết :

- › **delete** để xoá một cặp key - value bên trong object.

```
main.js > ...
1 const student = {
2   name: "Alice",
3   age: 20
4 }
5 delete student.age
6 console.log(student) // {name: "Alice"}  
-
```

- › **Object.assign()** để gộp nhiều object thành một.

```
main.js > ...
1 const x = {name: "Alice"}
2 const y = {age: 20}
3 const z = Object.assign(x, y)
4 console.log(z) // {name: "Alice", age: 20}  
-
```

Bài tập thực hành 6: Sử dụng kiến thức về Object trong JS, bạn hãy thực hiện các yêu cầu sau:

- › Viết một đoạn chương trình JavaScript tạo ra một object student có các thuộc tính sau:
 - name: Tên học sinh.
 - age: Tuổi học sinh.
 - hobbies: Mảng chứa sở thích của học sinh.
- › Viết một đoạn chương trình JavaScript để in ra tất cả các thuộc tính của đối tượng **student** ở yêu cầu trên.
- › Viết một đoạn chương trình JavaScript để xoá thuộc tính **age** của đối tượng **student** ở yêu cầu đầu tiên.
- › Viết một đoạn chương trình JavaScript để kiểm tra xem thuộc tính **name** có tồn tại trong đối tượng **student** ở câu 1 hay không.



Function

Ngoài Array - Object ra, một khái niệm vô cùng quan trọng mà chúng ta sẽ tìm hiểu đó chính là Function. Function là một cách để giúp chúng ta đóng gói một đoạn code để có thể tái sử dụng nhiều lần. Code bên trong một function sẽ không được chạy ngay lập tức, mà chúng sẽ được chạy khi chúng ta “**gọi**” tới.

```
main.js > ...
1  function sayHello() {
2      // Block of code
3      console.log("Hello")
4      console.log("World")
5 }
```

Với việc khai báo một function như vậy, chúng ta có thể “**gọi**” function đó để chạy những câu lệnh bên trong:

```
main.js
1 sayHello()
```

Chúng ta có thể gọi một function nhiều lần để chạy lại nhiều lần đoạn code bên trong nó. Các function được khai báo như trên có tính chất hoisting. Tức là chúng có thể được sử dụng trước khi khai báo.

```
main.js > ...
1 sayHello()
2 function sayHello() {
3     // Code goes here ...
4 }
```

Một function có thể được chạy với các tham số đầu vào khác nhau. Các tham số này cho phép chúng nhận vào một giá trị ở thời điểm bắt đầu khởi chạy. Từ đó chúng có thể xử lý khác đi. Tham số truyền vào được nằm trong cặp ngoặc () của function.

```
main.js > ...
1 function sayHello(name) {
2   console.log("Hello " + name + "! Nice to meet you");
3 }
4 sayHello("Alice") // "Hello Alice! Nice to meet you"
5 sayHello("Bob") // "Hello Bob! Nice to meet you"
6
7 function sum(x, y) {
8   console.log(x, y)
9 }
10 sum(1, 2) // 3
11 sum(3, 4) // 7
```

Giá trị trả về

Một function khi được gọi có thể trả về một giá trị thông qua từ khoá return. Giá trị trả về đó sẽ được sử dụng cho các mục đích xử lý tiếp theo.

```
main.js > ...
1 function sum(a, b) {
2   return a + b
3 }
4
5 const x = sum(1, 2)
6 console.log(x) // 3
7
8 function toCelsius(fahrenheit) {
9   return (5/9) * (fahrenheit - 32);
10 }
11
12 const celsius = toCelsius(77)
13 console.log(celsius) // 25
..
```

Lưu ý với return

- › Nếu một function không có 'return', chúng sẽ trả về giá trị là 'undefined'.
- › Function sẽ bị dừng ngay lập tức khi gặp câu lệnh 'return'. Các dòng lệnh phía dưới sẽ không được gọi.



- › Chỉ có thể `return` một giá trị trong JavaScript. Tuy nhiên, chúng ta có thể trả về kết quả là một `array` hoặc `object` tùy ý.

Giống như trên, ở phần kiến thức này, chúng ta cũng cùng nhau thử thách bản thân bằng những bài tập, cũng như ôn lại kiến thức qua các bài tập này. Ở đây chúng ta có 3 bài tập như sau:

Bài thực hành 7: Tạo ra một Object có chứa toàn bộ thông tin về cầu thủ Fernando Torres, bao gồm tên, năm sinh, quốc tịch. (Vui lòng tham khảo tại Wikipedia hoặc những trang thông tin uy tín).

Hỏi người dùng có muốn nhập thêm thông tin không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y, hỏi tiếp người dùng muốn nhập thêm thông tin gì? giá trị cụ thể là gì?
- › Sau đó, thêm thông tin người dùng nhập vào Object ban đầu.
- › Nếu người dùng nhập N, in ra màn hình câu “**Chào tạm biệt**”.

Bài thực hành 8: In ra màn hình 1 Array chứa tên 5 loại trái cây: Cam, Xoài, Táo, Nho, Sầu Riêng.

Hỏi người dùng muốn nhập thêm trái cây nào không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y, hỏi tiếp tên trái cây muốn nhập, sau đó thêm vào Array ban đầu và in Array mới ra màn hình.
- › Nếu người dùng nhập N, in ra màn hình câu “**Cảm ơn**”.

Bài thực hành 9: In ra màn hình 1 Array chứa tên 5 loại trái cây : Cam, Xoài, Táo, Nho, Sầu Riêng.

Hỏi người dùng muốn xóa loại trái cây nào không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y, yêu cầu nhập vào vị trí và tên, sau đó kiểm tra vị trí I nhập vào có tồn tại không (check undefined).
 - Tồn tại thì xóa.
 - Không tồn tại thì báo lỗi và thoát.
- › Nếu người dùng nhập N, in ra màn hình câu “**Cảm ơn**”.

Khi đã nắm được các cú pháp và lý thuyết về JavaScript, câu hỏi tiếp theo

được đặt ra chính là : “JavaScript tương tác với HTML như thế nào? ”. Để trả lời cho câu hỏi này, chúng ta đi đến khái niệm DOM.

II. Ứng dụng ngôn ngữ Javascript (HTML/CSS) để lập trình ứng dụng đăng nhập hệ thống

Để lập trình một ứng dụng đăng nhập hệ thống sử dụng HTML/CSS và JavaScript, bạn có thể làm như sau:

Xây dựng giao diện đăng nhập bằng HTML/CSS: Bạn có thể sử dụng các thẻ HTML để tạo form đăng nhập, bao gồm các trường nhập liệu cho tên đăng nhập và mật khẩu. Sử dụng CSS để thiết kế giao diện cho form đăng nhập và các phần tử khác trên trang web.

Thêm JavaScript để kiểm tra thông tin đăng nhập: Khi người dùng nhập tên đăng nhập và mật khẩu vào form, JavaScript có thể được sử dụng để kiểm tra xem thông tin đó có hợp lệ hay không. Bạn có thể sử dụng các điều kiện để kiểm tra định dạng của tên đăng nhập và mật khẩu, và hiển thị thông báo lỗi nếu người dùng nhập thông tin không đúng.

Xử lý yêu cầu đăng nhập: Khi người dùng nhấn nút Đăng nhập, JavaScript có thể được sử dụng để gửi thông tin đăng nhập. Nếu thông tin đăng nhập hợp lệ sẽ hiển thị thông báo đăng nhập thành công và ngược lại.

```
<form>
  <label for="username">Tên đăng nhập:</label>
  <input type="text" id="username" name="username">

  <label for="password">Mật khẩu:</label>
  <input type="password" id="password" name="password">

  <button type="button" onclick="validateLogin()">Đăng nhập</button>
</form>
```

CSS:

```
form {
  display: flex;
  flex-direction: column;
```



```
    align-items: center;
}

input {
    margin: 10px 0;
    padding: 5px;
    width: 200px;
}

button {
    margin-top: 20px;
    padding: 10px;
    background-color: blue;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}
```

JavaScript:

```
function validateLogin() {
    const username = document.getElementById("username").value;
    const password = document.getElementById("password").value;

    if (username === "" || password === "") {
        alert("Vui lòng nhập tên đăng nhập và mật khẩu");
        return;
    }

    if (username.length < 5 || password.length < 5) {
        alert("Tên đăng nhập và mật khẩu phải có ít nhất 5 ký tự");
        return;
    }
}
```

III. Lời giải

Bài tập thực hành 1: Tạo một trang hiển thị thông báo:

“Tôi là bậc thầy múa Florentino”

Để tạo một trang hiển thị thông báo bằng JavaScript, bạn có thể sử dụng phương thức alert() của đối tượng window. Cụ thể, để hiển thị thông báo “Tôi là bậc thầy múa Florentino”, bạn có thể sử dụng mã sau:

```
alert("Tôi là bậc thầy múa Florentino");
```

Sau đó, bạn có thể chèn mã này vào trong thẻ **<script>** trong tệp HTML của trang web của mình:

```
<!DOCTYPE html>
<html>
<head>
    <title>Thông báo bằng JavaScript</title>
</head>
<body>

    <!-- Thẻ chứa mã JavaScript -->
    <script>
        alert("Tôi là bậc thầy múa Florentino");
    </script>

</body>
</html>
```

Sau khi lưu lại và mở tệp HTML này trên trình duyệt web, thông báo “Tôi là bậc thầy múa Florentino” sẽ xuất hiện.

Bài tập thực hành 2: Cùng với bài trên, chúng ta hãy trích xuất nội dung code của bạn vào một file ngoài “**main.js**” và nằm trong cùng một thư mục.

Đầu tiên, bạn tạo một file có tên: Main.js. Sau đó viết đoạn code như bài trên:



main.js

```
1 alert("Tôi là bậc thầy múa Florentino");
2
```

Sau đó, bạn có thể chèn mã này vào thẻ `<script>` trong tệp HTML của trang web của mình:

```
5 index1.html > ...
1 <html>
2 <head>
3   <title>First JS application</title>
4 </head>
5 <body>
6   ...
7   <script src="./main.js"></script>
8 </body>
9 </html>
10
```

Bài thực hành 3: Giá trị cuối cùng của tất cả các biến a, b, c và d sau đoạn code dưới đây là bao nhiêu?

```
let a = 1, b = 1;
let c = ++a; : ?
let d = b++; : ?
```

Sau khi thực thi đoạn mã, giá trị cuối cùng của các biến là: a = 2, b = 2, c = 2, d = 1.

Giải thích:

- › a ban đầu được khởi tạo bằng 1. Sau đó, toán tử `++` được sử dụng với biến a trong biểu thức `++a`. Toán tử `++` trước biến sẽ tăng giá trị của biến lên 1 và sau đó trả về giá trị mới đó. Do đó, a sẽ được tăng lên 1 và giá trị của c sẽ được gán bằng 2.
- › b cũng ban đầu được khởi tạo bằng 1. Tuy nhiên, toán tử `++` được sử dụng với biến b sau biến a trong biểu thức `b++`. Toán tử `++` sau biến sẽ trả về giá trị hiện tại của biến trước khi tăng giá trị lên 1. Do đó, d sẽ được gán bằng giá trị hiện tại của b (1) và sau đó b sẽ được tăng lên 1, giá trị cuối cùng của b sẽ là 2.

Bài tập thực hành 4: Yêu cầu người dùng nhập một số 'n', sau đó:

- ✓ In ra số chẵn, lẻ < n.

```
main.js > ...
1 // Yêu cầu người dùng nhập một số n
2 let n = parseInt(prompt("Nhập số n: "));
3
4 // In ra số chẵn, lẻ < n
5 let evenNumbers = [];
6 let oddNumbers = [];
7
8 for (let i = 1; i < n; i++) {
9     if (i % 2 === 0) {
10         evenNumbers.push(i);
11     } else {
12         oddNumbers.push(i);
13     }
14 }
15 console.log(`Các số chẵn nhỏ hơn ${n}: ${evenNumbers}`);
16 console.log(`Các số lẻ nhỏ hơn ${n}: ${oddNumbers}`);
```

- ✓ Tìm ra số chia hết cho 7 lớn nhất nhỏ hơn n.

```
main.js > ...
1 // Tìm số chia hết cho 7 lớn nhất nhỏ hơn n
2 let maxMultipleOf7 = 0;
3
4 for (let i = n - 1; i > 0; i--) {
5     if (i % 7 === 0) {
6         maxMultipleOf7 = i;
7         break;
8     }
9 }
10 console.log(`Số chia hết cho 7 lớn nhất nhỏ hơn ${n}: ${maxMultipleOf7}`);
11
12
```

- ✓ Tính tổng các số đếm từ 1 đến n: $1+2+3+\dots+n$.

```
main.js > ...
1 // Tính tổng các số đếm từ 1 đến n
2 let sum = 0;
3
4 for (let i = 1; i <= n; i++) {
5     sum += i;
6 }
7
8 console.log(`Tổng các số từ 1 đến ${n}: ${sum}`);
9
```



- › Kiểm tra n có phải là số nguyên tố không.

```
main.js > ...
1 // Kiểm tra n có phải số nguyên tố không
2 let isPrime = true;
3
4 if (n === 1) {
5   isPrime = false;
6 } else {
7   for (let i = 2; i <= Math.sqrt(n); i++) {
8     if (n % i === 0) {
9       isPrime = false;
10      break;
11    }
12  }
13}
14
15 console.log(`$n là số ${isPrime ? "nguyên tố" : "không nguyên tố"}`);
16
```

- › In ra hình vuông dấu * với cạnh là n (vuông đặc và rỗng).

Hình vuông đặc có cạnh là n:

```
main.js > ...
1 let n = parseInt(prompt("Nhập vào số n: "));
2 let square = "";
3
4 for (let i = 1; i <= n; i++) {
5   for (let j = 1; j <= n; j++) {
6     square += "*";
7   }
8   square += "\n";
9 }
10
11 console.log(square);
```

Hình vuông rỗng có cạnh là n:

```
main.js > ...
1 let n = parseInt(prompt("Nhập vào số n: "));
2 let square = "";
3
4 for (let i = 1; i <= n; i++) {
5   for (let j = 1; j <= n; j++) {
6     if (i == 1 || i == n || j == 1 || j == n) {
7       square += "*";
8     } else {
9       square += " ";
10    }
11  }
12  square += "\n";
13 }
14
15 console.log(square);
16
```

- In ra tam giác cân với cạnh đáy có độ dài là n (các cạnh là dấu *).

```
main.js > ...
1 let n = parseInt(prompt("Nhập vào số n: "));
2 let triangle = "";
3
4 for (let i = 1; i <= n; i++) {
5   for (let j = 1; j <= n - i; j++) {
6     triangle += " ";
7   }
8   for (let k = 1; k <= 2 * i - 1; k++) {
9     triangle += "*";
10  }
11  triangle += "\n";
12}
13
14 console.log(triangle);
15
```

Bài tập thực hành 5: Dựa vào kiến thức đã học về Array, bạn hãy giải quyết các yêu cầu sau:

- Viết một hàm sum nhận vào một mảng các số nguyên và trả về tổng của chúng.

```
main.js > ...
1 function sum(arr) {
2   let total = 0;
3   for (let i = 0; i < arr.length; i++) {
4     total += arr[i];
5   }
6   return total;
7 }
8
9 // Sử dụng hàm sum để tính tổng của mảng numbers
10 const numbers = [1, 2, 3, 4, 5];
11 const total = sum(numbers);
12 console.log(total); // Output: 15
```

- Viết một hàm findMax nhận vào một mảng các số nguyên và trả về giá trị lớn nhất trong mảng đó.



```

main.js > ...
1  function findMax(arr) {
2    let max = arr[0];
3    for (let i = 1; i < arr.length; i++) {
4      if (arr[i] > max) {
5        max = arr[i];
6      }
7    }
8    return max;
9  }
10
11 // Sử dụng hàm findMax để tìm giá trị lớn nhất trong mảng numbers
12 const numbers = [1, 5, 3, 2, 4];
13 const max = findMax(numbers);
14 console.log(max); // Output: 5

```

- Viết một hàm removeDuplicates nhận vào một mảng các chuỗi và trả về một mảng mới loại bỏ các phần tử trùng lặp.

```

main.js > ...
1  function removeDuplicates(arr) {
2    const result = [];
3    for (let i = 0; i < arr.length; i++) {
4      if (!result.includes(arr[i])) {
5        result.push(arr[i]);
6      }
7    }
8    return result;
9  }
10
11 // Sử dụng hàm removeDuplicates để loại bỏ các phần tử trùng lặp trong mảng fruits
12 const fruits = ["apple", "banana", "orange", "apple", "kiwi", "banana", "banana"];
13 const uniqueFruits = removeDuplicates(fruits);
14 console.log(uniqueFruits); // Output: ["apple", "banana", "orange", "kiwi"]
15

```

- Viết một hàm reverseArray nhận vào một mảng và trả về một mảng mới với các phần tử được đảo ngược vị trí.

```

main.js > ...
1  function reverseArray(arr) {
2    const result = [];
3    for (let i = arr.length - 1; i >= 0; i--) {
4      result.push(arr[i]);
5    }
6    return result;
7  }
8
9  // Sử dụng hàm reverseArray để đảo ngược mảng numbers
10 const numbers = [1, 2, 3, 4, 5];
11 const reversedNumbers = reverseArray(numbers);
12 console.log(reversedNumbers); // Output: [5, 4, 3, 2, 1]
13

```

Bài thực hành 6: Sử dụng kiến thức về Object trong JS, bạn hãy thực hiện các yêu cầu sau:

- Viết một đoạn chương trình JavaScript tạo ra một object student có các thuộc tính sau:
 - name: Tên học sinh.
 - age: Tuổi học sinh.
 - hobbies: Mảng chứa sở thích của học sinh.

```
main.js > ...
1 let student = {
2   name: "John",
3   age: 20,
4   hobbies: ["music", "movies", "sports"]
5 };
```

- Viết một đoạn chương trình JavaScript để in ra tất cả các thuộc tính của đối tượng student ở yêu cầu trên.

```
main.js > ...
1 for (let key in student) {
2   console.log(key + ":" + student[key]);
3 }
```

- Viết một đoạn chương trình JavaScript để xóa thuộc tính age của đối tượng student ở yêu cầu đầu tiên.

```
main.js
1 delete student.age;
2
```

- Viết một đoạn chương trình JavaScript để kiểm tra xem thuộc tính name có tồn tại trong đối tượng student ở câu 1 hay không.

```
main.js
1 if ("name" in student) {
2   console.log("name is a property of student");
3 }
```



Bài thực hành 7: Tạo ra một Object có chứa toàn bộ thông tin về cầu thủ Fernando Torres, bao gồm tên, năm sinh, quốc tịch. (Vui lòng tham khảo tại Wikipedia hoặc những trang thông tin uy tín).

Hỏi người dùng có muốn nhập thêm thông tin không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y , hỏi tiếp người dùng muốn nhập thêm thông tin gì ? giá trị cụ thể là gì ?
- › Sau đó, thêm thông tin người dùng nhập vào Object ban đầu.
- › Nếu người dùng nhập N, in ra màn hình câu “**Chào tạm biệt**”.

```
main.js > ...
1 // Tạo Object với thông tin ban đầu
2 let fernandoTorres = {
3   ten: 'Fernando Torres',
4   namSinh: '20/03/1984',
5   quocTich: 'Tây Ban Nha',
6 };
7
8 // Hỏi người dùng có muốn nhập thêm thông tin không
9 let answer = prompt('Bạn có muốn nhập thêm thông tin không? (Y/N)');
10
11 while (answer === 'Y') {
12   let info = prompt('Thông tin muốn thêm vào là gì?');
13   let value = prompt('Giá trị của thông tin ${info} là gì?');
14
15   // Thêm thông tin người dùng nhập vào Object ban đầu
16   fernandoTorres[info] = value;
17
18   answer = prompt('Bạn có muốn nhập thêm thông tin không? (Y/N)');
19 }
20
21 if (answer === 'N') {
22   console.log('Chào tạm biệt');
23 }
```

Bài thực hành 8: In ra màn hình 1 Array chứa tên 5 loại trái cây: Cam, Xoài, Táo, Nho, Sầu Riêng.

Hỏi người dùng muốn nhập thêm trái cây nào không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y , hỏi tiếp tên trái cây muốn nhập, sau đó thêm vào Array ban đầu và in Array mới ra màn hình.
- › Nếu người dùng nhập N, in ra màn hình câu “**Cảm ơn**”.

```

main.js > ...
1 // Tạo một Array chứa tên 5 loại trái cây
2 let fruits = ["Cam", "Xoài", "Táo", "Nho", "Sầu Riêng"];
3
4 // Hỏi người dùng có muốn nhập thêm trái cây không
5 let userInput = prompt("Bạn có muốn nhập thêm trái cây vào danh sách không? (Y/N)");
6
7 // Nếu người dùng muốn nhập thêm trái cây
8 if (userInput === "Y") {
9     // Hỏi tên trái cây muốn nhập
10    let newFruit = prompt("Hãy nhập tên trái cây muốn thêm vào danh sách:");
11
12    // Thêm tên trái cây vào Array
13    fruits.push(newFruit);
14
15    // In Array mới ra màn hình
16    console.log(fruits);
17 }
18 // Nếu người dùng không muốn nhập thêm trái cây
19 else if (userInput === "N") {
20     // In ra câu "Cảm ơn"
21     console.log("Cảm ơn!");
22 }
23 // Nếu người dùng nhập sai định dạng
24 else {
25     // In ra câu "Định dạng không hợp lệ"
26     console.log("Định dạng không hợp lệ");
27 }

```

Chú ý rằng lời giải này chỉ sử dụng `prompt` để yêu cầu người dùng nhập liệu, không có kiểm tra đầu vào để đảm bảo định dạng đúng hoặc hợp lệ. Trong thực tế, cần kiểm tra và xử lý các đầu vào khác nhau để đảm bảo ứng dụng hoạt động chính xác và an toàn.

Bài thực hành 9: In ra màn hình 1 Array chứa tên 5 loại trái cây: Cam, Xoài, Táo, Nho, Sầu Riêng.

Hỏi người dùng muốn xóa loại trái cây nào không bằng cách nhập Y hoặc N.

- › Nếu người dùng nhập Y, yêu cầu nhập vào vị trí và tên, sau đó kiểm tra vị trí | nhập vào có tồn tại không (check undefined).
 - Tồn tại thì xóa.
 - Không tồn tại thì báo lỗi và thoát.
- › Nếu người dùng nhập N, in ra màn hình câu “**Cảm ơn**”.

```

main.js > ...
1 let fruits = ["Cam", "Xoài", "Táo", "Nho", "Sầu Riêng"];
2
3 console.log("Danh sách trái cây: " + fruits);
4
5 let response = prompt("Bạn có muốn xóa một loại trái cây không? (Y/N)").toLowerCase();
6
7 if (response === "y") {
8     let position = Number(prompt("Nhập vị trí của trái cây muốn xóa:"));
9     if (position >= 0 && position < fruits.length) {
10         let name = prompt("Nhập tên trái cây muốn xóa:");
11         if (fruits[position] === name) {
12             fruits.splice(position, 1);
13             console.log("Danh sách trái cây sau khi xóa: " + fruits);
14         } else {
15             console.log("Không tìm thấy trái cây cần xóa tại vị trí đã nhập!");
16         }
17     } else {
18         console.log("Vị trí vừa nhập không tồn tại trong danh sách trái cây!");
19     }
20 } else {
21     console.log("Cảm ơn!");
22 }

```

Giải thích cho bài trên như sau:

- › Đầu tiên, chúng ta tạo một mảng fruits chứa tên của 5 loại trái cây. Sau đó, in danh sách trái cây này ra màn hình sử dụng console.log().
- › Chúng ta sử dụng hàm prompt() để hỏi người dùng có muốn xóa một loại trái cây không.
- › Nếu người dùng nhập "Y", chúng ta yêu cầu người dùng nhập vào vị trí của trái cây muốn xóa và tên của nó. Sau đó, kiểm tra vị trí vừa nhập có nằm trong phạm vi của mảng hay không bằng cách kiểm tra xem nó có lớn hơn hoặc bằng 0 và nhỏ hơn độ dài của mảng hay không.
- › Nếu vị trí vừa nhập hợp lệ, chúng ta kiểm tra xem tên trái cây có trùng với tên được nhập hay không. Nếu có, chúng ta sử dụng phương thức splice() để xóa phần tử tại vị trí đó ra khỏi mảng và in ra mảng mới sau khi đã xóa sử dụng console.log().
- › Nếu không tìm thấy trái cây cần xóa tại vị trí đã nhập, chúng ta sử dụng console.log() để in ra thông báo lỗi.
- › Nếu vị trí vừa nhập không hợp lệ, chúng ta sử dụng console.log() để in ra thông báo lỗi.
- › Nếu người dùng nhập "N", chúng ta sử dụng console.log() để in ra thông báo "Cảm ơn".