Introduction

Ba Nguyễn

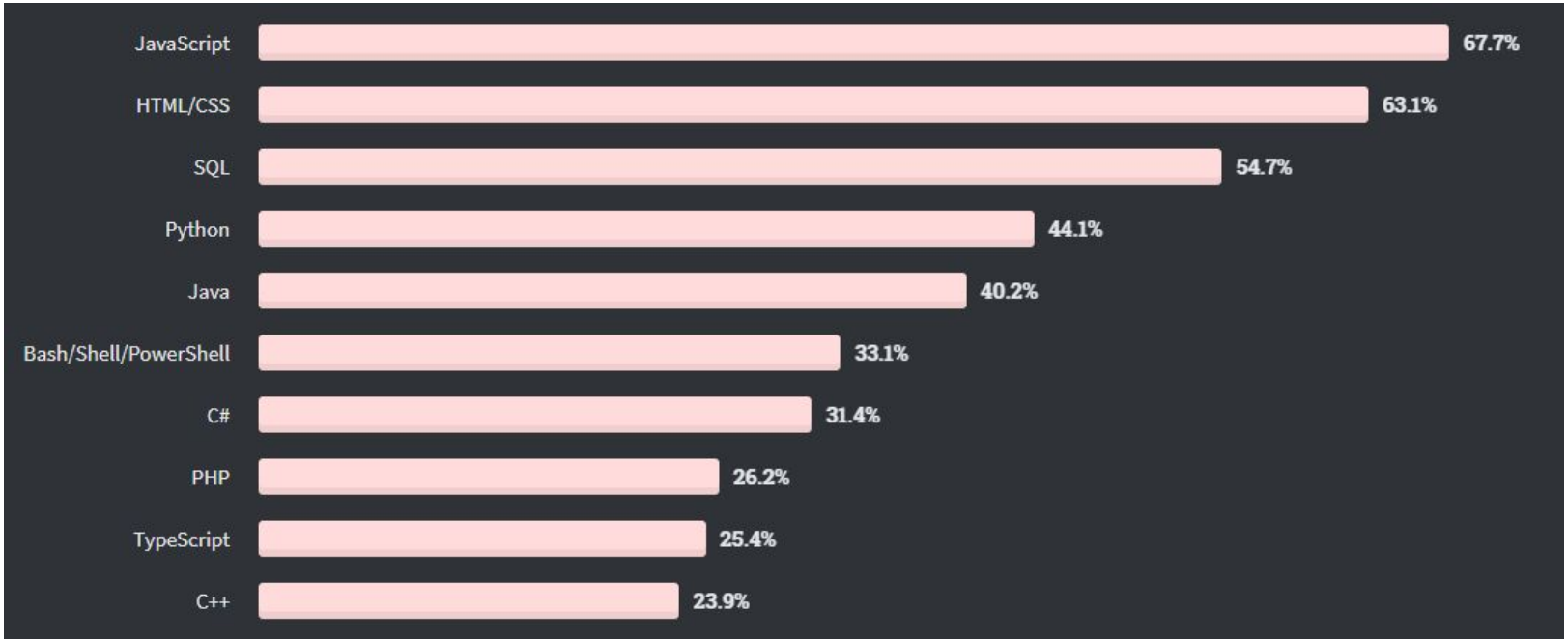
What is JavaScript?

JavaScript là một “ngôn ngữ lập trình kịch bản” (scripting language), ban đầu, nó được tạo ra với mục đích duy nhất - cung cấp tính tương tác cho các trang web.

Ngày nay, JavaScript là ngôn ngữ phổ biến và có tốc độ phát triển nhanh nhất, nó được sử dụng trong nhiều mục đích và không còn bị giới hạn trong trình duyệt mà có thể sử dụng ở rất nhiều nền tảng / môi trường khác nhau, như máy chủ, điện thoại, ...

Trong môi trường trình duyệt, các đoạn mã JavaScript có thể được nhúng trong trang HTML và chạy tự động khi trang web được tải

What is JavaScript?

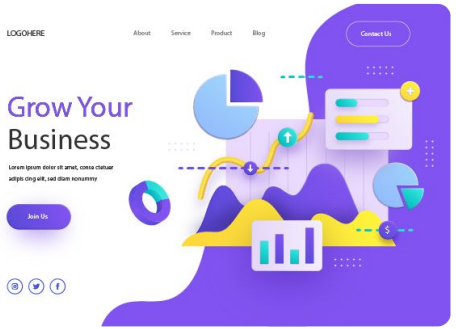


Top 10 ngôn ngữ phổ biến nhất 2020 (stackoverflow)

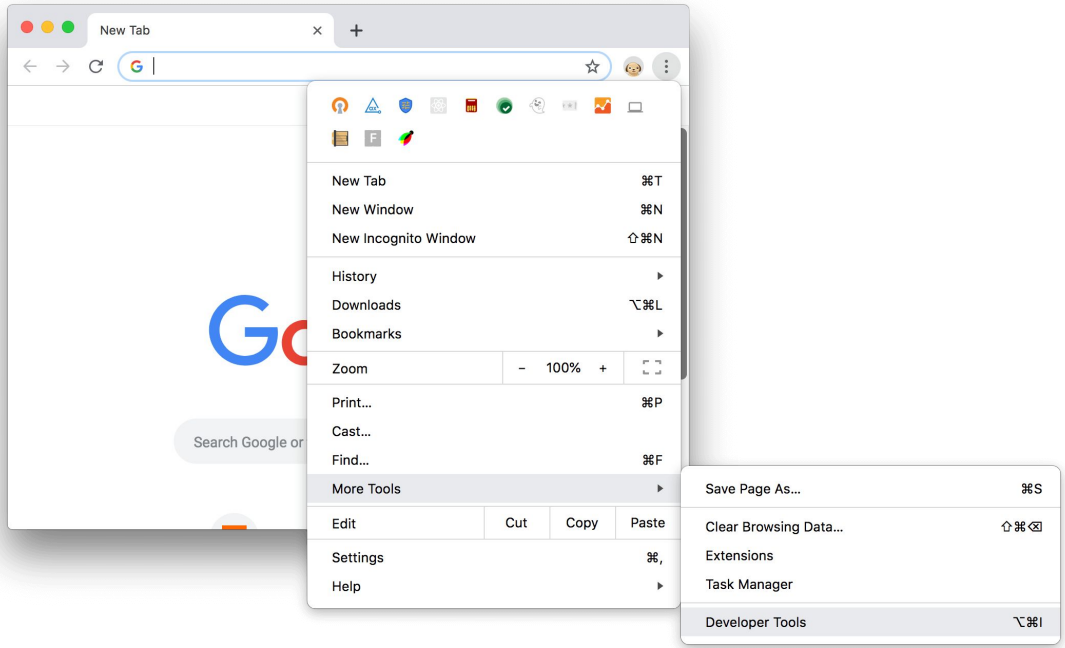
What can you do with JavaScript?

JavaScript ngày càng mạnh mẽ và có thể làm được rất nhiều thứ như:

- Web / Mobile Apps
- Real-time Networking Apps
- Command-line Tools
- Games
- ...



Dev Tools



Basic

Ba Nguyễn

Hello World

Nhúng mã JavaScript vào trang web

```
<body>
    ...
    <!-- Inline JS -->
    <h1 onclick="alert('Hello JS 😉')">Heading</h1>

    <!-- Internal JS -->
    <script>
        alert("Hello JS 😉");
    </script>

    <!-- External JS -->
    <script src="script.js"></script>
</body>
```

Hello World

Statements

```
alert("Hello JS 😉");
alert("💡 Mỗi câu lệnh nên kết thúc với ; ");
```

Comments

```
// Oneline comment
/*
Multi-line comments
💡 Chú thích giúp nhớ và hiểu cách mã thực thi
*/
```

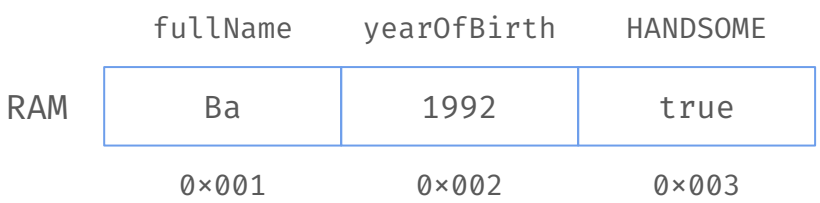
Variables

Biến là yếu tố cơ bản của bất kỳ ngôn ngữ lập trình nào. Biến là *tên một vùng nhớ lưu trữ dữ liệu trong bộ nhớ máy tính.*

Cú pháp khai báo biến trong JavaScript:

```
let fullName = "Ba Nguyễn";
let yearOfBirth = 1992;
const HANDSOME = true;
var myPower = 99.99;
let a = b = c = "😛";
var d = "😂",
    e = "😴";

alert(firstName); // Ba
```



Variables

`let`

Biến khai báo với `let` không thể khai báo lại, tuy nhiên, giá trị của biến có thể thay đổi

```
let myName = "Ba";
let myName = "Oops"; // ❌ Error
myName = "Ba Nguyễn"; // 👌 Okey
myName = 123; // 👌 Okey
myName = true; // 👌 Okey
```

`const`

Biến khai báo với `const` (hằng số) không thể khai báo lại, và cũng không thể thay đổi giá trị

```
const MYNAME = "Ba";
const MYNAME = "Oops"; // ❌ Error
MYNAME = "Ba Nguyễn"; // ❌ Error
MYNAME = 123; // ❌ Error
MYNAME = true; // ❌ Error
```

Variables

`var` là cách khai báo biến cũ, biến được khai báo với `var` được phép khai báo lại, thay đổi giá trị, và phạm vi tồn tại - *scope* - của biến cũng khác biệt so với `let` và `const`.

```
var myName = "Ba";
var myName = "Ba Nguyễn"; // 👌 Okey
myName = "Bar"; //👌 Okey
myName = 3; //👌 Okey
```

💡 Nên sử dụng `let` và `const` để khai báo biến

Variables Naming Rules & Conventions

Quy tắc đặt tên biến

- Tên biến **chỉ được chứa** *ký tự*, *số*, hoặc ký tự đặc biệt `$` và `_`
- Tên biến **không được** bắt đầu bằng *một số*
- Tên biến **có phân biệt** *chữ hoa*, *chữ thường*
- Tên biến **không được** trùng với *từ khóa* của JavaScript

Quy ước đặt tên biến

- JavaScript sử dụng phong cách `camelCase` cho tên biến, hoặc hàm
- Với hằng số - `const` - sử dụng `UPPERCASE`

Exercise

Tên biến nào không hợp lệ?

```
let 3a = 1;
let var = 10;
let my-name = "Ba Nguyễn";
const job = "Developer";
const PI = 3.14;
var __ = "$$";
var let = "__";
```

```
let x5 = 55;
let x_X = "OmG";
let $y__ = 1;
const ___ = "GOOD";
var PI = 3.14;
let FIRSTNAME = "Ba";
const birthday = "24.05.1992";
let LAstNaME = "Nguyễn";
```

Data Types

Ba Nguyễn

Data Types

Có **9** kiểu dữ liệu trong JavaScript

1. `number`
2. `bigint`
3. `string`
4. `boolean`
5. `undefined`
6. `null`
7. `symbol`
8. `object`
9. `function`

Primitives	Reference
`number`	`object`
`bigint`	`array`
`string`	`function`
`boolean`	
`undefined`	
`null`	
`symbol`	

💡 JavaScript là ngôn ngữ có kiểu *động* (*dynamic typing* - hoặc *yếu* - *weakly typing*), không cần phải khai báo kiểu dữ liệu cho biến trước khi sử dụng, và có thể thay đổi giá trị của biến thành bất kỳ kiểu dữ liệu nào

Numbers

Bao gồm cả số nguyên và số thực, giới hạn `-2^53 + 1` đến `2^53 - 1`

```
let integer = 1;
let float = 1.2345;
let complex = 1e10;
let binary = 0b01;
let hex = 0xff;
```

Các giá trị đặc biệt trong kiểu `number`

```
Infinity; // 1 / 0
-Infinity; // -1 / 0
NaN; // Not a Number
```

Strings

Là một chuỗi ký tự được đặt trong cặp dấu ‘ ’, “ ”

```
let firstName = "Ba";
let lastName = "Nguyễn";

let greeting = "Hi, I'm " + firstName + " " + lastName;
alert(greeting); // Hi, I'm Ba Nguyễn

let message = 'I\'m very "handsome"'; // escape string
alert(message); // I'm very "handsome"

// Một số ký tự đặc biệt (escape) \' \" \\ \n \t
```

Booleans, Null, Undefined

Kiểu `boolean` hay *logic* chỉ bao gồm hai giá trị `true` hoặc `false`

```
let isPretty = true;
let isUgly = false;
```

`null` và `undefined` là 2 giá trị đặc biệt, `null` đại diện cho một đối tượng *không tồn tại*, còn `undefined` đại diện cho một đối tượng *chưa được gán giá trị*.

```
let selectedColor = null;
let age = undefined;
```

💡 Khi khai báo một biến mà không gán giá trị, biến sẽ có giá trị là `undefined`

```
let noValue; // undefined
```

Object

`object` là kiểu dữ liệu đặc biệt, các biến với kiểu dữ liệu khác chỉ lưu trữ được một giá trị tại một thời điểm, còn `object` cho phép lưu cùng lúc nhiều giá trị trong một biến duy nhất, các dữ liệu được lưu trong `object` có thể thuộc bất kỳ kiểu nào

```
let ba = {
    name: "Ba",
    age: 29,
    handsome: true,
};
// Dot Notation
alert(ba.name); // Ba
// Bracket Notation
alert(ba["age"]); // 29
```

Array

`array` là cấu trúc dữ liệu cơ bản trong JavaScript, `array` cho phép lưu trữ một danh sách các giá trị/đối tượng ***có thứ tự*** và cung cấp các các cách thức đặc biệt để xử lý các giá trị trong nó

```
let colors = ["red", "green", "blue"];
// Bracket Notation
alert(colors[0]); // red
alert(colors[1]); // green
```

Function

`function` là cách thức tổ chức mã cơ bản trong JavaScript, `function` là một tập hợp các câu lệnh để xử lý một nhiệm vụ hoặc tính toán một giá trị, nó đóng gói các câu lệnh trong một khối mã cho phép tái sử dụng các đoạn mã nhiều lần

```
// Function Declaration
function greeting() {
    alert("Hello JavaScript");
}

// Call Function
greeting(); // Hello JavaScript
```

Function

`function` có thể nhận các giá trị đầu vào - *input* - và thay đổi cách nó hoạt động dựa trên giá trị đó, mỗi `function` có thể nhận số lượng giá trị đầu vào bất kỳ.

```
// Parameter
function greeting(name) {
    alert("Hello " + name);
}

// Argument
greeting("Ba"); // Hello Ba
greeting("Bon"); // Hello Bon
```

Two Types of Function

```
// Thực hiện một công việc
function greeting(name) {
    alert("Hello " + name);
}

// Tính toán và trả về một kết quả (giá trị)
function add(a, b) {
    return a + b;
}
add(1, 2); // 3
```

💡 Mặc định mọi hàm luôn trả về một giá trị - `undefined`

Interactions

JavaScript cung cấp sẵn một số hàm để tương tác (nhập, hiển thị dữ liệu) trên trình duyệt

```
// Hiển thị bảng thông báo
alert("Hello");
// Hiển thị bảng nhập dữ liệu, trả về string hoặc null
let yourName = prompt("What's your name?");
// Hiển thị bảng chọn yes/no, trả về true | false
let answer = confirm("Do you love me, " + yourName + "?");
// In giá trị ra console
console.log(answer);
```

Type Conversions

JavaScript ***tự động chuyển đổi kiểu dữ liệu*** khi thực hiện tính toán các giá trị, đồng thời cung cấp thêm một số phương thức để chuyển đổi từ kiểu dữ liệu này sang kiểu dữ liệu khác

```
let result = 1 + 2; // 3
let answer = "1 + 2 = " + result;
// result từ "number" → "string"
// "1 + 2 = 3"

let other = 1 - "1"; // 0
// "1" từ "string" → "number"
```

Type Conversions

```
// String
String(123); // "123"
String(true); // "true"
String(null); // "null"
String(undefined); // "undefined"
String(); // ""
String("Ba Nguyễn"); // "Ba Nguyễn"
```

Type Conversions

```
// Number
Number("123.456"); // 123.456
Number("-123.456"); // -123.456

// Một số giá trị đặc biệt khi chuyển về kiểu number
Number(undefined); // NaN
Number(null); // 0
Number(true); // 1
Number(false); // 0
Number("  123   "); // Khoảng trắng được loại bỏ → 123
Number("123abc"); // Chuỗi chứa ký tự → NaN
Number(""); // Chuỗi rỗng → 0
```

Type Conversions

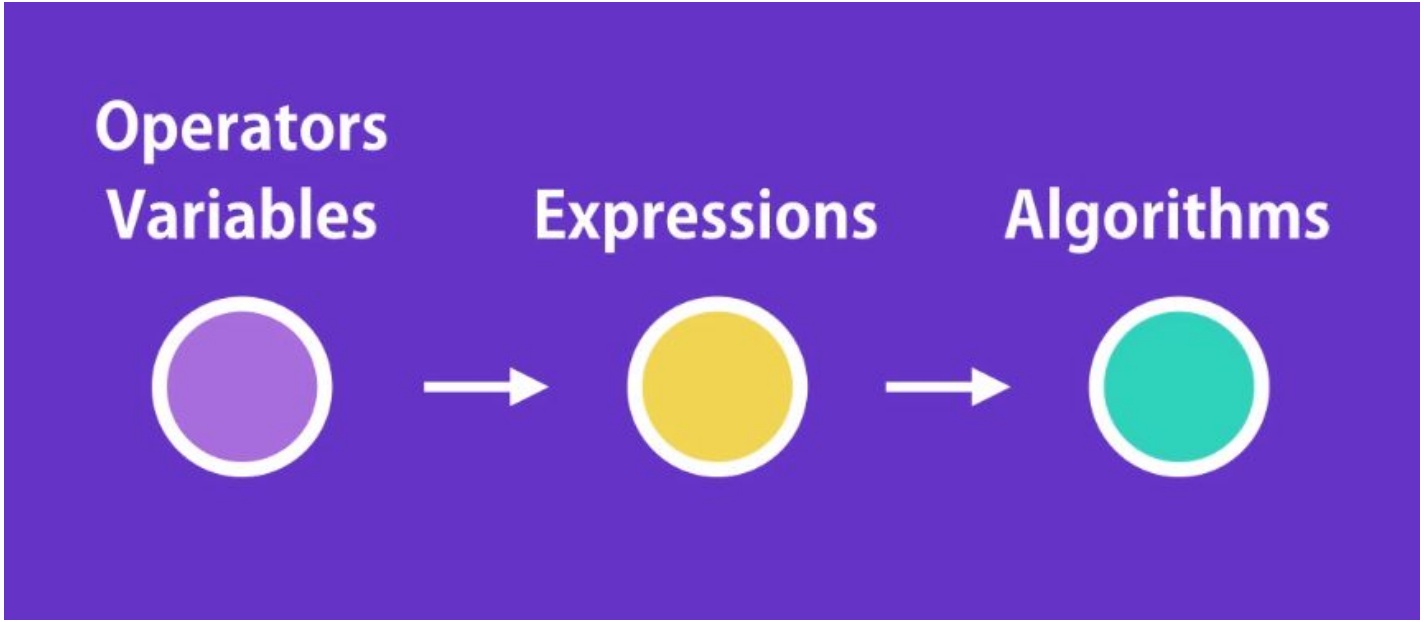
```
// Boolean
Boolean(123); // true
Boolean("Ba đẹp trai"); // absolutely true
Boolean("Ba xấu trai"); // absolutely false

// Một số giá trị đặc biệt khi chuyển về kiểu boolean
"", 0, null, undefined, NaN; // false
// Tất cả các giá trị khác → true
```

Operators

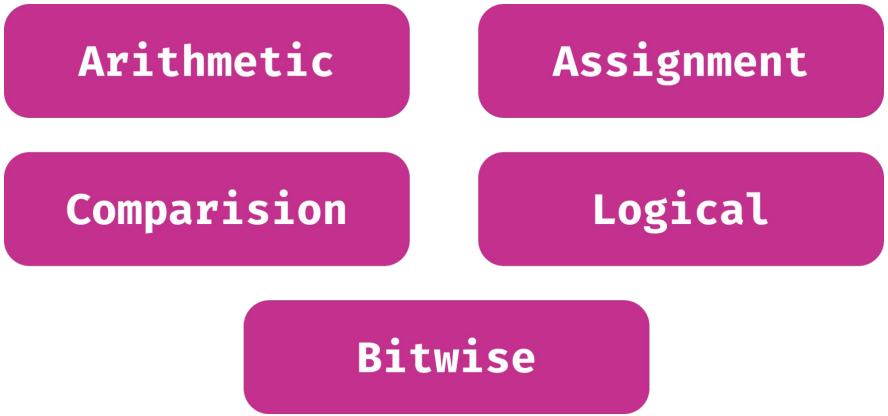
Ba Nguyễn

Operators



Operators

Toán tử trong JavaScript được chia thành 5 loại:



💡 Quy tắc thực hiện biểu thức theo thứ tự *từ trái qua phải, dựa theo độ ưu tiên toán tử*. Các toán tử có độ ưu tiên khác nhau, quyết định phép tính nào sẽ được thực hiện trước

Arithmetic Operators

Đối với kiểu dữ liệu `string` phép `+` chuyển đổi kiểu dữ liệu của toán hạng về kiểu `string` và thực hiện nối chuỗi

```
1 + "2"; // "12"
1 + 2 + "3"; // "33"
"1" + 2 + true; // "12true"
```

Phép nối chuỗi ***chỉ hoạt động duy nhất với toán tử*** `+` , với những toán tử số học khác, mọi kiểu dữ liệu được chuyển về `number`

```
6 - "2" + null - true; // 3
"5" / 3 + 2 / "1"; // 4
```

Arithmetic Operators

Basic operator: `+ - * / % **`

```
// Chia lấy phần dư %
5 % 3; // 2
"6" % 2; // 0
1 % true; // 0

// Lũy thừa **
2 ** 2; // 4
2 ** "2"; // 4
2 ** false; // 1

// Mọi phép tính (trừ phép + chuỗi) với NaN đều cho kết quả là NaN
5 - NaN / 2; // NaN
1 * 2 * undefined; // NaN
```

Exercise

1. Viết chương trình nhập 2 số `a`, `b`, tính và in ra kết quả tất cả phép tính cơ bản với 2 số đó
2. Viết chương trình nhập vào chiều dài, chiều rộng của hình chữ nhật, tính và in ra chu vi, diện tích của hình chữ nhật đó
3. Viết chương trình nhập vào bán kính hình tròn, tính và in ra chu vi, diện tích của hình tròn

Homework

1. Viết chương trình nhập vào một giá trị đo độ dài ở đơn vị *cm*, tính và in ra giá trị tương ứng ở các đơn vị *mm*, *m*, *km*
2. Viết chương trình nhập vào một giá trị nhiệt độ ở đơn vị *Celsius*, in ra nhiệt độ ở đơn vị *Fahrenheit* và *Kevin* tương ứng
3. Viết chương trình nhập thời gian hiện tại *tính theo số giây* từ 0h, tính và in ra giá trị giờ/phút/giây tương ứng theo định dạng `h:m:s`

Assignment Operators

```
// Gán =
let a = 3; // a = 3
let b = 1 + 2 + 3; // b = 6
let c = (a = b + 5); // b = 6, a = 11, c = 11

// Gán kết hợp
a += b; // a = a + b
a /= b; // a = a / b
a %= b + 5; // a = a % (b + 5)
a *= a + 2; // a = a * (a + 2);
```

Asignment Operators

Increment, Decrement (tự tăng/giảm)

`++` và `--` là hai toán tử đặc biệt, nó thực hiện phép tính ***tăng/giảm*** giá trị của biến đi `1`, hai toán tử này có thể đặt ở trước biến - *prefix* hoặc sau biến - *postfix*.

Khi đặt trên một câu lệnh riêng biệt thì không có sự khác nhau

```
let a = 1;
a++; // a = 2
a--; // a = 1
++a; // a = 2
--a; // a = 1
```

Asignment Operators

Increment, Decrement (tự tăng/giảm)

Tuy nhiên, khi đặt trong một biểu thức, *postfix* - tăng/giảm `a` đi `1` và trả về ***giá trị trước đó (giá trị trước khi tăng/giảm)***, *prefix* - cũng tăng/giảm `a` đi `1` nhưng trả về ***giá trị mới (giá trị sau khi tăng/giảm)***

```
let a = 1;
let b = a++ + 2; // a = 2, b = 1 + 2 = 3
let c = ++a + 2; // a = 3, c = 3 + 2 = 5

let d = a++ + ++a - a-- - --a;
// a = 3, d = 3 + 5 - 5 - 3 = 0
```

Exercise

Tính giá trị các biểu thức

```
let a = 1,
    b = (a % 2) * 2,
    c = a++ - b--,
    d = "0";

a + b + c + d;
a - b + c - d;
a-- + (b-- * c) / d;
++a - +b * c + d;
d + ++a + (--b % c);
a-- - +d++ - ++c + b--;
a++ - b-- + ++c + --d;
```

```
let a = 1,
    b = (a * 2) / 2,
    c = a-- + b++,
    d = "-0";

a - b - c - d;
a + b - c + d;
a++ - (b++ / c) * d;
--a + -b / c - d;
d - --a - ++b * c;
a++ + -d-- + --c - b++;
a-- + b++ - --c - ++d;
```

Comparision Operators

Toán tử so sánh `==` `≠` `>` `≥` `<` `≤` `===` `≠=`

💡 Kết quả của các phép so sánh là một giá trị `boolean`

`==` `≠` `>` `≥` `<` `≤` tự động chuyển đổi kiểu dữ liệu của 2 toán hạng về cùng một kiểu và thực hiện so sánh

```
2 < 3; // true
2 ≥ 3; // false
2 == "2"; // true
2 ≠ "2"; // false
1 > null; // null → 0 → true
2 ≤ "2"; // true
```

Comparision Operators

Toán tử so sánh `==` `!=` `>` `>=` `<` `<=` `===` `!==`

`===` và `!==` (*strict comparison*) so sánh cả kiểu giá trị của dữ liệu

```
2 === 2; // number vs number và 2 == 2 → true
2 === "2"; // number vs string → false
2 !== "2"; // number vs string → true
2 !== 2; // number vs number → false
0 === null; // number vs object → false
```

Comparision Operators

So sánh chuỗi

JavaScript sử dụng bảng mã Unicode, khi so sánh 2 chuỗi, nó thực hiện so sánh từng ký tự dựa theo thứ tự trong bảng mã (Unicode table)

```
"a" > "A"; // true
"A" > "Z"; // false
"Ba" == "Ba"; // true
"Ba" ≤ "Ba Nguyễn"; // true
```

💡 Tham khảo bảng mã Unicode: wiki/unicode_character

Comparision Operators

`null`, `undefined`, `NaN`

`null` và `undefined` là 2 trường hợp đặc biệt

```
null == 0; // false
null ≤ 0; // true
null ≥ 0; // true
null == undefined; // true
null === undefined; // false
```

💡 Mọi biểu thức so sánh với `NaN` đều cho kết quả là `false`

Logical Operators

Toán tử logic `||` - or, `&&` - and, `!` - not

`||` - `or` tìm giá trị `true` đầu tiên trong biểu thức (chuyển về kiểu `boolean`) và trả về giá trị đó, nếu không có giá trị nào là `true` thì trả về giá trị cuối cùng trong biểu thức (bất kể `true` hay `false`)

```
true || false; // true
0 || 1; // 1
0 || false || ""; // ""
"abc" || true; // "abc"
let age = 16;
age > 18 || alert("Bạn chưa đủ tuổi"); // ?
```

Logical Operators

Toán tử logic `||` - or, `&&` - and, `!` - not

`&&` - `and` tìm giá trị `false` đầu tiên trong biểu thức (chuyển về kiểu `boolean`) và trả về giá trị đó, nếu không có giá trị nào là `false` thì trả về giá trị cuối cùng trong biểu thức (bất kể `true` hay `false`)

```
true && false; // false
0 && 1; // 0
0 && false && ""; // 0
"abc" && true; // true
let age = 16;
age > 18 && alert("Bạn đủ tuổi rồi :)"); // ?
```

Logical Operators

Toán tử logic `||` - or, `&&` - and, `!` - not

`!` - `not` chuyển giá trị về kiểu `boolean` và phủ định nó

```
!""; // true
!"123"; // false
!!false; // false
!!"xxx"; // true
!!!!!!!!!!!!!!false; // false
```

Exercise

Tính giá trị các biểu thức

```
let a = true,
    b = !a;
let c = (!a && !!b) || 0;

a && b && c;
a || b || c;
(a && !b) || !!c;
!(a || !b) && c;
!!(a && !!b) || !c;
```

```
let a = false,
    b = !!a;
let c = a || (!b && 0);

a && b && c;
a || b || c;
!a || (b && !c);
!!(!a && b) || c;
!(!a || !b) || c;
```

Control Statements

Ba Nguyễn

If Else

Thực thi các câu lệnh theo điều kiện chỉ định, điều kiện có thể là một giá trị, biểu thức, hàm, ... và được tự động chuyển đổi về kiểu `boolean`.

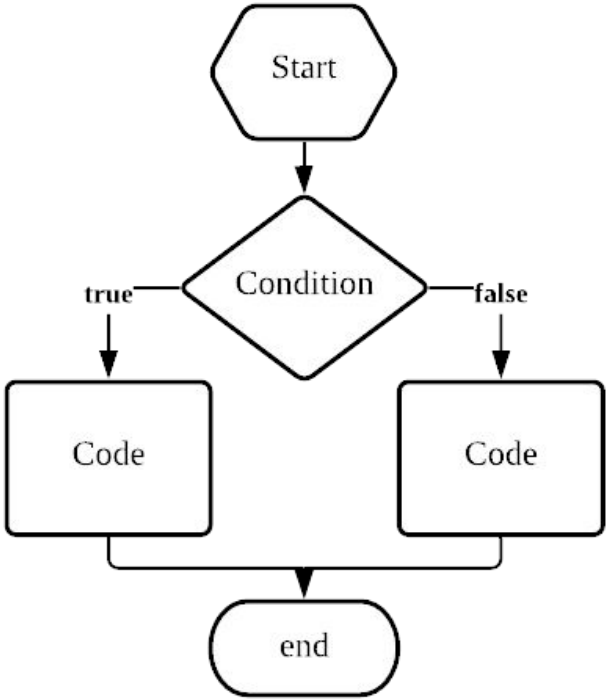
Mệnh đề `if` chỉ định điều kiện và mã bên trong chỉ được thực thi nếu điều kiện đúng, ngược lại, mệnh đề `else` sẽ được thực thi

Cú pháp:

```
if (condition) {
    // run if condition is true
} else {
    // run if condition is false
}
```

If Else

```
if (condition) {
    // condition
} else if (otherCondition) {
    // other condition
} else if (anotherCondition) {
    // another condition
} else {
    // if all condition false
}
```



If Else

```
let number = Number(prompt("Enter a number"));

if (number > 10) {
    console.log("Your greater than 10");
} else {
    console.log("Your number is less than 10");
}
```

Exercise

1. Viết chương trình nhập vào một số `a`, kiểm tra số đó là chẵn hay lẻ và in kết quả
2. Viết chương trình nhập vào 2 số `a`, `b`, kiểm tra và in ra số lớn hơn
3. Viết chương trình nhập vào một tháng trong năm, in ra mùa tương ứng
4. Viết chương trình nhập một số `a`, kiểm tra và in ra số đó có chia hết cho cả 5 và 11 hay không
5. Viết chương trình nhập vào 2 số `a`, `b` là tham số của phương trình bậc nhất `ax + b = 0`, tính và in ra nghiệm của phương trình đó

Homework

1. VCT nhập ba số `a`, `b`, `c`, kiểm tra và in ra số lớn nhất
2. VCT nhập một năm `year`, kiểm tra và in ra năm đó có phải năm nhuận hay không
3. VCT nhập một ký tự `char`, kiểm tra và in ra ký tự đó là nguyên hay phụ âm (tiếng Anh)
4. VCT nhập một ký tự `char`, kiểm tra và in ra ký tự đó là chữ thường hay chữ in hoa
5. VCT nhập ba hệ số `a`, `b`, `c`, của phương trình bậc 2 $ax^2 + bx + c = 0$, tính và in ra nghiệm phương trình đó
6. VCT nhập số điểm `point` ở thang điểm 10 của sinh viên quy đổi sang thang điểm chữ:
   - `point` $\leqslant$ `10` $\Rightarrow$ `A`
   - `point` `<` `8.5` $\Rightarrow$ `B`
   - `point` `<` `7.0` $\Rightarrow$ `C`
   - `point` `<` `5.5` $\Rightarrow$ `D`
   - `point` `<` `4.0` $\Rightarrow$ `F`

?

Toán tử `?` (toán tử hỏi, toán tử ba ngôi) là cú pháp rút gọn của `if else`, thường dùng khi muốn gán một giá trị theo điều kiện

Cú pháp:

```
condition ? value1 : value2;
```

```
condition
    ? value1
    : otherCondition
        ? value2
        : anotherCondition;
```

```
let yearOld = prompt("How old are u?");

let age =
    yearOld < 15
        ? "Kid"
        : yearOld < 18
            ? "Teen"
            : "Adult";
```

Exercise

💡 Làm lại **Exercise** của `if else` sử dụng toán tử `?`

For loops

Thực hiện lặp lại một hành động (khối mã) cho tới khi điều kiện trở thành `false`

Cú pháp:

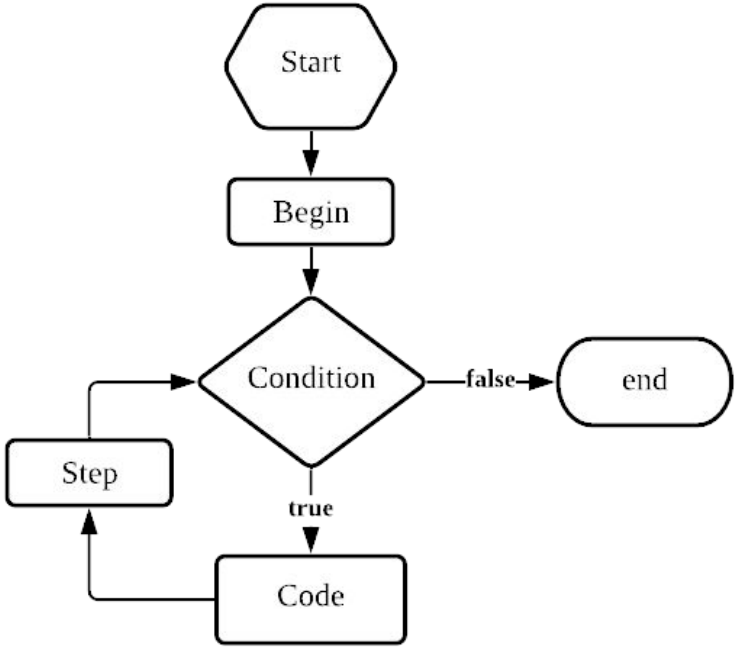
```
for (begin; condition; step) {
    // code
}
// begin: khởi tạo (các) giá trị ban đầu
// condition: (các) điều kiện của vòng lặp
// step: cập nhật (các) giá trị sau mỗi vòng lặp
```

For loops

```
console.log("Before loop");

for (let i = 0; i < 10; i++) {
    console.log(i);
}

console.log("After loop");
// Before loop
// 1 2 3 4 5 6 7 8 9 10
// After loop
```



For loops

```
// Tính tổng từ 1 đến 10
let sum = 0;

for (let i = 1; i ≤ 10; i++) {
    sum += i;
}

console.log(sum); // 55
```

For loops

Các bước thực hiện vòng lặp `for`

1. Khởi tạo biến `i = 1` (chỉ khởi tạo 1 lần duy nhất)
2. Kiểm tra `i <= 10`
3. Tính `sum += i`
4. Tăng `i` (`i++`), lúc này `i = 2`
5. Quay trở về bước 2, kiểm tra `i <= 10`
6. Lại tính `sum += i`
7. Lại tăng `i` (`i++`), lúc này `i = 3`
8. Lặp lại cho tới khi điều kiện `i <= 10` trở thành `false` (khi `i = 11`)
9. Kết thúc vòng lặp

Exercise

1. Viết chương trình tính tổng từ `0 → 100` và in ra kết quả
2. Viết chương trình tính tổng tất cả số lẻ trong khoảng `0 → 100` và in ra kết quả
3. Viết chương trình tính tổng tất cả các *số lẻ chia hết* cho `3` trong khoảng `0 → 100` và in ra kết quả
4. Viết chương trình tính tổng tất cả các số *chia hết cho cả* `3`, `5` và `7` trong khoảng `0 → 100` và in ra kết quả
5. Viết chương trình tính tổng *bình phương của tất cả các số chia hết* cho `3` hoặc `5` hoặc `7` trong khoảng `0 → 100` và in ra kết quả

Break, continue

Câu lệnh `break` ***ngay lập tức*** ngắt vòng lặp hiện tại, đồng thời các câu lệnh bên dưới bị bỏ qua

```
for (let i = 0; i < 10; i++) {
    if (i == 2) {
        break;
    }

    // nếu i == 2, bỏ qua câu lệnh này
    console.log(i);
    // đồng thời ngắt vòng lặp
}
```

Câu lệnh `continue` dừng lần lặp hiện tại, chuyển lớp lần lặp tiếp theo, các câu lệnh bên dưới cũng bị bỏ qua

```
for (let i = 0; i < 10; i++) {
    if (i == 2) {
        continue;
    }

    // nếu i == 2 bỏ qua câu lệnh này
    console.log(i);
    // chuyển tới vòng lặp tiếp i == 3
}
```

Inner loops

Các vòng lặp `for` (hay các cấu trúc điều khiển khác như `if else`) có thể sử dụng lồng nhau để giải quyết những bài toán phức tạp hơn

```
for (let i = 0; i < 5; i++) {
    for (let j = i + 1; j < 5; j++) {
        console.log(`i = ${i}, j = ${j}`);
    }
}

if (age < 18) {
    if (withParent) { /* Okela */ }
    else { /* Noooooooo */ }
} else { /* Okela */ }
```

Label

`break` và `continue` chỉ hoạt động với vòng lặp hiện tại (*vòng lặp mà nó được đặt*), để sử dụng với vòng lặp cấp cao hơn (*bên ngoài*), sử dụng *nhãn - lable*

```
outerLoop: for (let i = 0; i < 5; i++) {
    innerLoop: for (let j = i + 1; j < 5; j++) {
        if (i > 3) {
            break outerLoop;
        }

        console.log(`i = ${i}, j = ${j}`);
    }
}
```

Loops

💡 Cần chú ý điều kiện dừng, tránh vòng lặp vô hạn

💡 Vòng lặp `for` có thể bỏ qua một, hoặc tất cả biểu thức bên trong `( )`:

```
let i = 0;
for (; i < 10; i++) {
// code
}
```

```
let i = 0;
for (;;) {
    // code, condition, step
}
```

💡 `break`, `continue` chỉ hoạt động với `if else`, không hoạt động với toán tử `?`

```
for (let i = 0; i < 10; i++) {
    (i % 2 == 0) ? break; false; // ❌ error
}
```

Homework

1. VCT tính và in bảng cửu chương, sử dụng vòng lặp `for` lồng nhau
2. VCT in ra nếu số *chia hết* cho `3` thì in `Fizz`, *chia hết* cho `5` thì in `Buzz`, *chia hết cho cả* `3` và `5` thì in `FizzBuzz`, *không chia hết cho cả* `3` và `5` thì in `BizzFuzz`, trong khoảng `0 → 50`
3. VCT tính và in ra tổng bội chung của `3` và `5` trong khoảng `0 → 100`
4. VCT nhập vào một số `n`, kiểm tra số đó có phải số nguyên tố hay không và in ra kết quả
5. VCT nhập vào 2 số `a`, `b` kiểm tra và in ra các số nguyên tố trong khoảng `a → b`
6. VCT in ra bảng cửu chương ngược (từ `10 → 1`)

Homework

1. VCT in ra chữ số đầu và cuối của một số. VD `12345 → 15`
2. VCT kiểm tra và in ra một số có phải số `Palindrome` hay không
3. VCT kiểm tra và in ra một số có phải số `Armstrong` hay không
4. VCT tính và in ra giai thừa `Factorial` của một số
5. VCT kiểm tra và in ra một số có phải số `Perfect` hay không
6. VCT kiểm tra và in ra một số có phải số `Strong` hay không