



VIETNAM
AUSTRALIA
Vocational College

LẬP TRÌNH JAVASCRIPT CƠ BẢN

TỔNG QUAN VỀ JAVASCRIPT



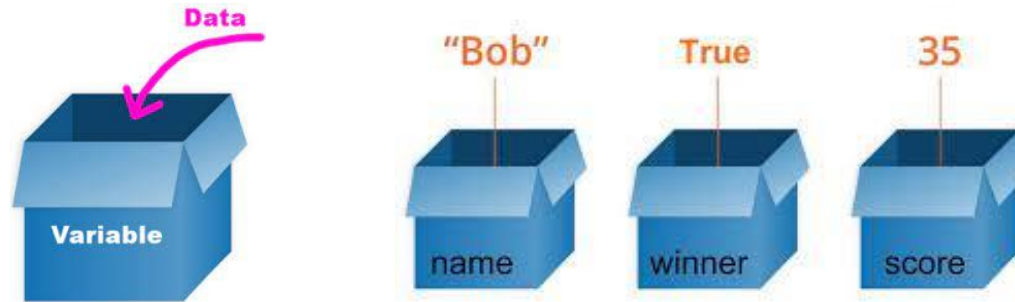
Nội dung:

1. Biến (variables) & hằng (constants)
2. Giá trị (values) & kiểu (types)
3. Biểu thức (expressions) & toán tử (operators)
4. Khối lệnh (blocks)
5. Các lệnh cơ bản (statements)
6. Lưu đồ thuật toán (flowchart)



Biến (variables) & hằng (constants)

- ❑ **Biến** là một biểu tượng (giống như là một thùng chứa) có tên dùng để lưu trữ một giá trị dữ liệu.



- ❑ Có 2 kiểu giá trị phổ biến trong các ngôn ngữ lập trình:
 - **Static typing** (kiểu tĩnh): **Tách biệt về kiểu dữ liệu của biến.**
 - **Dynamic typing** (kiểu động): **Linh động về kiểu dữ liệu của biến.**
- ❑ **Biến trong JavaScript** là **kiểu Dynamic typing**.
 - **Khai báo biến** sử dụng từ khóa **var** (không có thông tin kiểu cụ thể khi khai báo biến)



Biến (variables) & hằng (constants)

❑ Ví dụ:

➤ Biến **amount** ban đầu giữ một number **99.99**, và sau đó giữ kết quả **number**, kết quả của **amount * 2** là **199.98**.

➤ Lệnh **console.log(..)** đầu tiên ngầm buộc giá định **number** thành **string** để in ra.

➤ Tiếp đó lệnh **amount = "\$" + String(amount)** rõ ràng ép giá trị **199.98** thành một **string** và thêm ký tự **"\$"** đằng trước. Tại đây, **amount** có **string** giá trị **"\$199.98"**, vì thế **console.log(..)** thứ hai không cần phải cưỡng ép gì cả để in nó ra.

```
JS variable.js X
JS variable.js > ...
1  var amount = 99.99;
2  amount = amount * 2;
3  console.log(amount); // 199.98
4
5  // chuyển `amount` sang một string,
6  // và thêm "$" ở đầu
7  amount = "$" + String(amount);
8  console.log(amount); // "$199.98";
```



Biến (variables) & hằng (constants)

❑ Ví dụ:

➤ Lập trình viên JavaScript sẽ lưu ý tính linh hoạt của việc sử dụng biến **amount** cho các giá trị **99.99.199.98** và **“\$199.98”**.

➤ Những người thích **static-typing** sẽ thích tách biến kiểu như **amountStr** để giữ giá trị **“\$199.98”** cuối cùng, bởi vì nó là một kiểu khác.

➤ Để ý, bạn sẽ thấy **amount** giữ một giá trị đang chạy mà thay đổi trong quá trình hoạt động của chương trình.

```
JS variable.js X
JS variable.js > ...
1  var amount = 99.99;
2  amount = amount * 2;
3  console.log(amount); // 199.98
4
5  // chuyển `amount` sang một string,
6  // và thêm "$" ở đầu
7  amount = "$" + String(amount);
8  console.log(amount); // "$199.98";
```



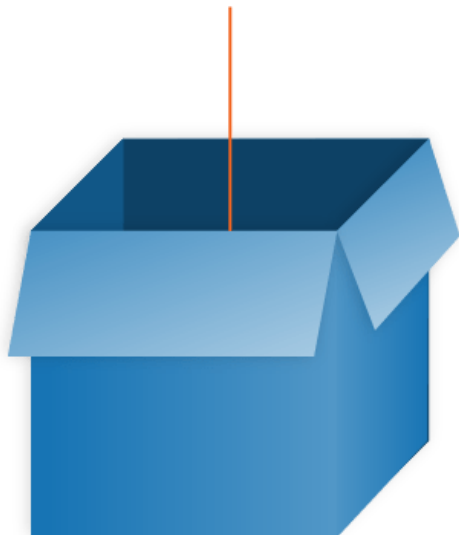
Biến (variables) & hằng (constants)

❑ Mục đích của biến:

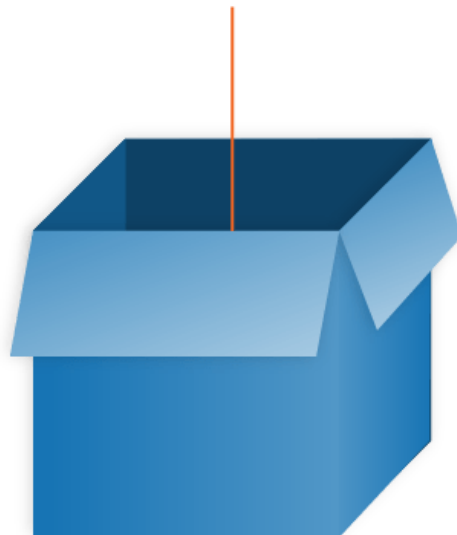
➤ Quản lý **state** của chương trình. **State** là **theo dõi các thay đổi của giá trị khi chương trình hoạt động**.

➤ Ví dụ: **var box = "Bob"; -> box = true; -> box = 35;**

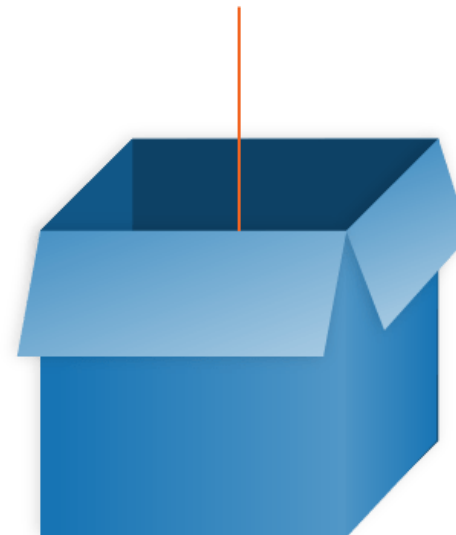
"Bob"



true



35



```
> var box = "Bob";  
< undefined  
  
> box  
< "Bob"  
  
> box = true;  
< true  
  
> box  
< true  
  
> box = 35;  
< 35  
  
> box  
< 35
```



Biến (variables) & hằng (constants)

- ❑ Một cách sử dụng phổ biến khác của các biến là **tập trung thiết lập giá trị tại một chỗ**, gọi là hằng (constants).
- ❑ Hằng được sử dụng khi bạn **muốn khai báo một biến với một giá trị xác định** và **giá trị này không thay đổi trong suốt chương trình**.

```
JS constant.js X
JS constant.js > ...
1  var TAX_RATE = 0.08; // 8% sales tax
2  var amount = 99.99;
3  amount = amount * 2;
4  amount = amount + (amount * TAX_RATE);
5  console.log(amount); // 215.9784
6  console.log(amount.toFixed(2)); // "215.98"
```

The screenshot shows a VS Code terminal window with a dark theme. The title bar at the top says 'JS constant.js' with a close button. The terminal prompt is 'JS constant.js > ...'. The code is written in a monospaced font with syntax highlighting: keywords like 'var' and 'console.log' are in blue, strings and comments are in green, and numbers and operators are in white. The code calculates a tax amount. The bottom of the terminal shows tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL' (which is active). To the right of the tabs are icons for 'cmd', a plus sign, a down arrow, an up arrow, and a close button.



Biến (variables) & hằng (constants)

- ❑ **Hằng** thường được khai báo ở đầu chương trình, tiện để bạn có một chỗ để thay đổi giá trị khi cần.
- ❑ Theo quy ước, các biến Hằng thường được viết hoa, có gạch dưới _ giữa các liên từ.

```
JS constant.js X
JS constant.js > ...
1  var TAX_RATE = 0.08; // 8% sales tax
2  var amount = 99.99;
3  amount = amount * 2;
4  amount = amount + (amount * TAX_RATE);
5  console.log(amount); // 215.9784
6  console.log(amount.toFixed(2)); // "215.98"
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL cmd + v ^ X



Biến (variables) & hằng (constants)

❑ Ví dụ Hằng:

➤ Hàm **toFixed(..)** là **hàm giúp làm tròn số thập phân number** và **tạo ra string** như ý.

➤ Biến **TAX_RATE** là một **constant theo quy ước** (Không có gì đặc biệt ngăn biến này không thay đổi).

➤ Biến **TAX_RATE** giúp dễ dàng cập nhật lại chương trình (cấu hình lại **TAX_RATE**) ở một nơi, **thay vì phải tìm và cập nhật nhiều giá trị 0.08** nằm rải rác khắp chương trình.

```
JS constant.js X

JS constant.js > ...

1  var TAX_RATE = 0.08; // 8% sales tax
2  var amount = 99.99;
3  amount = amount * 2;
4  amount = amount + (amount * TAX_RATE);
5  console.log(amount); // 215.9784
6  console.log(amount.toFixed(2)); // "215.98"

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

D:\workspace\programming-techniques\lesson-02\lesson-02
215.9784
215.98

D:\workspace\programming-techniques\lesson-02\lesson-02
```



Biến (variables) & hằng (constants)

❑ Hằng trong ES6:

➤ Phiên bản mới thông dụng nhất của JS hiện nay là **ES6** đã có cách khai báo **constants**, bằng cách sử dụng **const** thay cho **var**.

➤ **Const** giống như **var** với giá trị không đổi.

➤ **Const** khác **var** ở chỗ sẽ ngăn ngừa sự thay đổi giá trị vô tình xảy ra ở đâu đó sau giá trị khởi tạo.

➤ Nếu khi thay đổi **TAX_RATE** sau lần khai báo đầu tiên **sẽ bị lỗi**.

```
JS constant_es6.js X
JS constant_es6.js > ...
1  const TAX_RATE = 0.08; // 8% sales tax
2  var amount = 99.99;
3  amount = amount * 2;
4  TAX_RATE = 0.09;
5  amount = amount + (amount * TAX_RATE);
6  console.log(amount); // 215.9784
7  console.log(amount.toFixed(2)); // "215.98"

PROBLEMS  OUTPUT  TERMINAL  ...
cmd + - ^ x

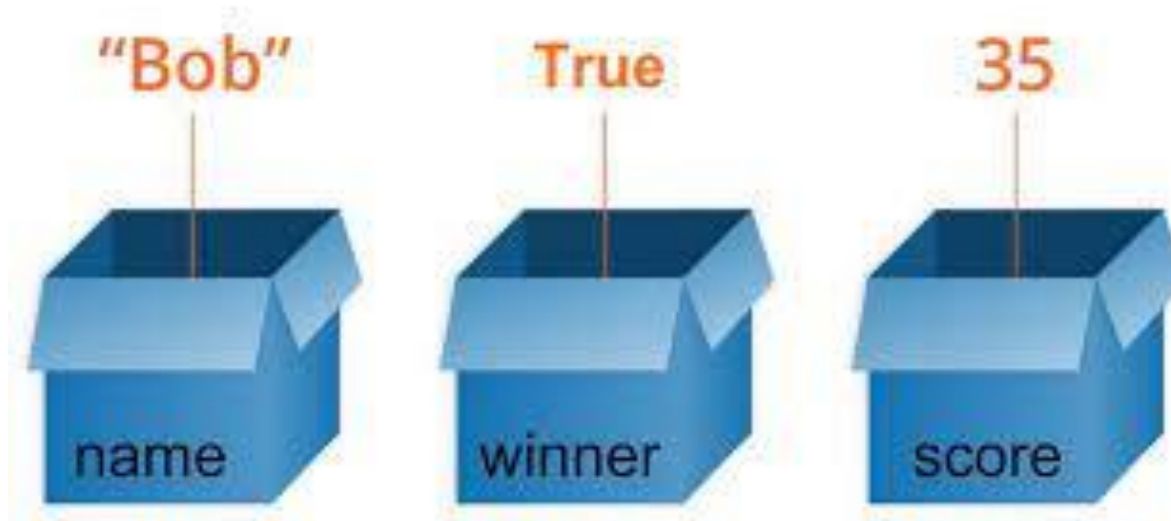
D:\workspace\programming-techniques\lesson-02\lesson-02-slides\node constant_es6.js
D:\workspace\programming-techniques\lesson-02\lesson-02-slides\constant_es6.js:4
TAX_RATE = 0.09;
    ^

TypeError: Assignment to constant variable.
    at Object.<anonymous> (D:\workspace\programming-techniques\lesson-02\lesson-02-slides\constant_es6.js:4:10)
```



Giá trị (values) & kiểu (types)

- ❑ Nếu bạn hỏi nhân viên tại một cửa hàng điện thoại **giá của một cái điện thoại gì đó**, họ sẽ trả lời **“chín chín, chín chín”** (Ví dụ: \$99.99) -> tức họ đã cho bạn một hình dung **giá trị tiền bạn cần phải trả**.
- ❑ Sau đó, bạn lại hỏi **điện thoại có đồ sạc hay không**, câu trả lời có thể là **“có”** hoặc **“không”** -> **giá trị đồ vật mà bạn có thể có hoặc không** có khi mua chiếc điện thoại đó.





Giá trị (values) & kiểu (types)

❑ Các **kiểu giá trị nguyên thủy** (primitive types):

- Khi bạn muốn làm toán, bạn muốn **number** (kiểu số). -> 15, 20
- Khi bạn muốn in giá trị trên màn hình, bạn cần **string** (một hay nhiều ký tự, từ, câu – kiểu chuỗi).
 - Các **giá trị được thêm trực tiếp vào source code** được gọi là **literals** (giá trị cố định – đã học)
 - **string** literals được bao bằng **dấu ngoặc kép “...”** hoặc **dấu ngoặc đơn ‘...’** -> chỉ là phong cách code khác nhau.
 - Khi bạn muốn tạo một quyết định trên chương trình, bạn cần **boolean (true hoặc false)** -> true, false
 - Ngoài, **string/number/boolean**, còn có **arrays, objects, functions,...**



Giá trị (values) & kiểu (types)



❑ Chuyển đổi giữa các kiểu dữ liệu (conversion):


➤ Nếu bạn có một **number** nhưng muốn hiển thị trên màn hình, bạn cần **chuyển đổi giá trị** thành **string**, và trong JavaScript gọi việc chuyển đổi này là “**ép kiểu**”.

▪ Ví dụ: ký tự số trên **form của trang bán hàng** là **string**, nhưng để sử dụng giá trị để tính toán, bạn cần ép kiểu thành **number**.

Chi tiết đơn hàng Chưa chuyển Nguồn Website

CHƯA GIAO

	Kem chanh nhiệt đới	35,010đ	×	2	70,020đ
	Trà hoa cúc	25,010đ	×	1	25,010đ
Ghi chú		Giá	95,030đ		
<input type="text" value="Thêm ghi chú cho đơn hàng"/>		Vận chuyển			
		Giao hàng tận nơi	20,000đ		
		0.00 kg			
		Tổng cộng	115,030đ		
		Khách hàng thanh toán	115,030đ		

 ĐƠN HÀNG ĐÃ XÁC NHẬN THANH TOÁN 115,030đ

Hoàn trả



Giá trị (values) & kiểu (types)

❑ Chuyển đổi giữa các kiểu dữ liệu (conversion):

➤ JavaScript **cung cấp** một số **hàm có sẵn để hỗ trợ ép kiểu**:

- **Number(...)**: function hỗ trợ ép bất kỳ kiểu khác sang **number**.
- Dấu bằng tương đối **==**: Ép kiểu ngầm định khi so sánh chuỗi số với số.

- Ví dụ: **"99.99" == 99.99**

- ♦ JavaScript sẽ ép kiểu ngầm định về trái từ chuỗi số thành số
"99.99" -> 99.99

- ♦ Sau đó, mới so sánh với vế phải (99.99)

- 99.99 == 99.99 -> true**

- **Lưu ý**: ép kiểu ngầm định rất dễ gây ra bug nếu không cẩn thận!



Biểu thức (expressions) & toán tử (operators)

□ Biểu thức (expressions):

- Trong toán học, **biểu thức** là một tập hợp gồm các chữ số và các phép toán.
- Trong biểu thức, các chữ số được gọi là **toán hạng**, các phép toán thì được gọi là **toán tử**.
- Ví dụ: **10 + 5 - 2** là một biểu thức, trong đó **10, 5, 2** là toán hạng, còn **+, -** là toán tử.
- Tương tự trong toán học, **biểu thức trong JavaScript** cũng là **tập hợp các toán hạng** và **toán tử**.



Biểu thức (expressions) & toán tử (operators)

❑ Toán tử (operators):

- **Toán tử** trong **JavaScript** có **chức năng giống toán tử trong toán học**.
- Tuy nhiên, **một vài toán tử** trong **JavaScript** có cách **viết khác so với cách viết toán tử trong toán học**.
- Danh sách **những toán tử cơ bản** trong JavaScript:

Toán tử	Tên gọi	Ví dụ	Kết quả
+	Phép cộng	10 + 4	14
-	Phép trừ	10 - 4	6
*	Phép nhân	10 * 4	40
/	Phép chia	10 / 4	2.5
%	Phép chia lấy phần số dư	10 % 4	2

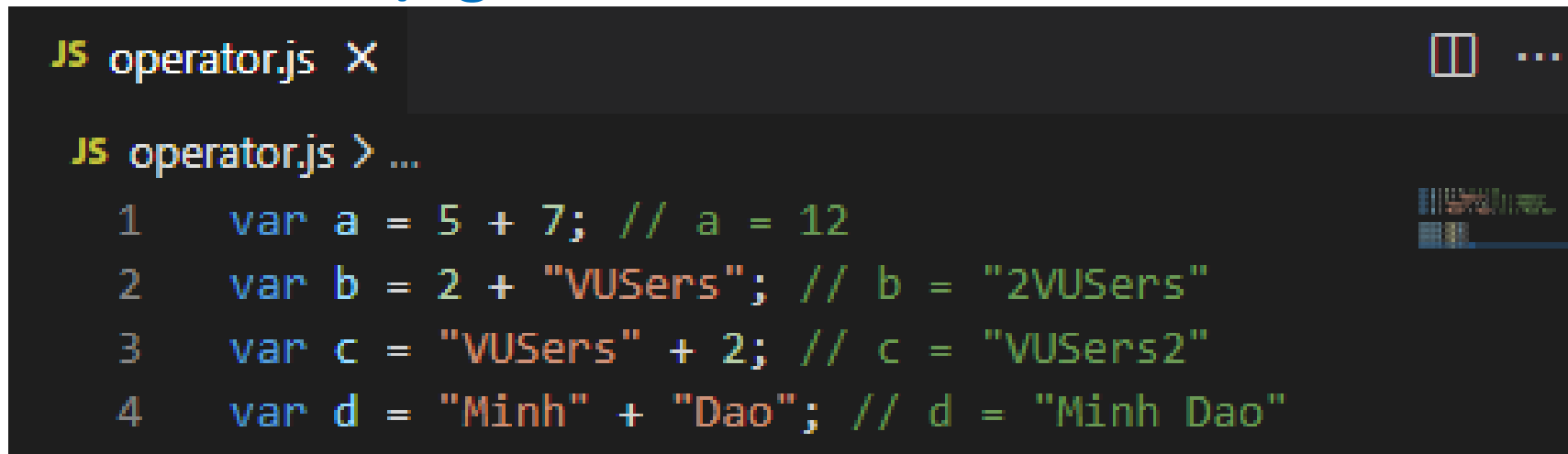


Biểu thức (expressions) & toán tử (operators)

❑ Toán tử (operators):

➤ **Phép cộng** trong JavaScript **tương đối khác** với phép cộng trong toán học. Trong JavaScript:

- Số có thể **cộng** số => cho **ra số**.
- Số có thể **cộng** chuỗi (hoặc chuỗi có thể **cộng** số) => cho **ra chuỗi**.
- Chuỗi có thể **cộng** chuỗi => cho **ra chuỗi**.



```
JS operator.js X
JS operator.js > ...
1  var a = 5 + 7; // a = 12
2  var b = 2 + "VUSers"; // b = "2VUSers"
3  var c = "VUSers" + 2; // c = "VUSers2"
4  var d = "Minh" + "Dao"; // d = "Minh Dao"
```



Biểu thức (expressions) & toán tử (operators)

❑ Toán tử (operators):

➤ Độ ưu tiên của toán tử:

Mức độ ưu tiên	Toán tử		
1	()		
2	*	/	%
3	+		-

➤ Ví dụ:

JS priority-operator.js ✕

JS priority-operator.js > ...

```
1 var result = 7 + 8 * (10 - 2) + 3 * 4;
```

```
2 console.log(result); // result = 83
```



Biểu thức (expressions) & toán tử (operators)

❑ Toán tử (operators):

➤ Ví dụ:

$7 + 8 * (10 - 2) + 3 * 4 \Rightarrow 7 + 8 * 8 + 3 * 4$

$7 + 8 * 8 + 3 * 4 \Rightarrow 7 + 64 + 3 * 4$

$7 + 64 + 3 * 4 \Rightarrow 7 + 64 + 12$

$7 + 64 + 12 \Rightarrow 71 + 12$

$71 + 12 \Rightarrow 83$

❑ Toán tử (operators):

➤ Khoảng trắng:

■ Trong JavaScript, dấu khoảng trắng giữa các toán hạng và toán tử **không ảnh hưởng đến kết quả phép toán**. Ví dụ: ba biểu thức dưới đây có cùng 1 kết quả.

```
JS space-operator.js X
JS space-operator.js > ...
1  var a = 7 + 8 - 3 * 4;
2  var b = 7 + 8 - 3 * 4;
3  var c = 7 + 8 - 3 * 4;
4  console.log(a); // 3
5  console.log(b); // 3
6  console.log(c); // 3
```



Khối lệnh (blocks)

- ❑ Nhân viên cửa hàng điện thoại phải **đi qua tất cả các khâu để hoàn tất việc thanh toán** khi bạn mua điện thoại.
- ❑ Tương tự, trong code chúng ta thường nhóm các biểu thức với nhau, thường được gọi là block.
- ❑ Trong JavaScript, một **block** được **xác định bằng cách bao một hoặc nhiều lệnh trong một cặp dấu ngoặc nhọn {...}** hello
- ❑ Ví dụ:

```
JS block.js  X
JS block.js > ...
1  var amount = 99.99;
2  // a general block
3  {
4      amount = amount * 2;
5      console.log(amount); // 199.98
6  }
```



Khối lệnh (blocks)

- ❑ Kiểu **block** `{..}` chung này **hợp lệ**, nhưng **không thường thấy trong các chương trình JS**.
- ❑ Một **block** **thường được gắn liền với một lệnh điều khiển** (if -> sẽ học sau).
- ❑ Ví dụ:

```
JS block-if.js X
JS block-if.js > ...
1  var amount = 99.99;
2  // is amount big enough?
3  if (amount > 10) { // <-- block attached to `if`
4      amount = amount * 2;
5      console.log(amount); // 199.98
6  }
```

- ❑ Chú ý: Một **block** lệnh **không cần dấu chấm phẩy (;) để kết thúc**.



Các lệnh cơ bản (statements)

❑ Có 6 lệnh JavaScript cơ bản cần biết:

- `console.log(..)`
- `alert(..)`
- `confirm(..)`
- `prompt(..)`

❑ Ngoài lệnh **`console.log(..)`**, các lệnh khác đều là hàm của đối tượng **`window`** (chỉ được khởi tạo trong **`browser`**), không hỗ trợ trong **`Node.js`**

❑ Do vậy, ta sẽ phải sử dụng **`DevTools`** của **`browser`** cho các bài toán cần **nhập xuất dữ liệu, in dữ liệu ra trình duyệt,...**



Các lệnh cơ bản (statements)

❑ Lệnh **console.log(..)**:

➤ Phương thức **console.log(..)** là phương thức **hỗ trợ viết một tin nhắn, đoạn văn bản ra console.**

➤ Cú pháp: **console.log(*message*)**

▪ *message*: Là một tin nhắn hoặc đối tượng được viết ra console.

➤ Ví dụ: in đoạn chữ “**Xin chào FastTracker**” ra màn hình console.

```
JS console-log.js X
JS console-log.js
1 console.log('Xin chào FastTracker');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL cmd + - ^ X

```
D:\workspace\programming-techniques\lesson-02\lesson-02-slides>node
console-log.js
Xin chào FastTracker
```



Các lệnh cơ bản (statements)

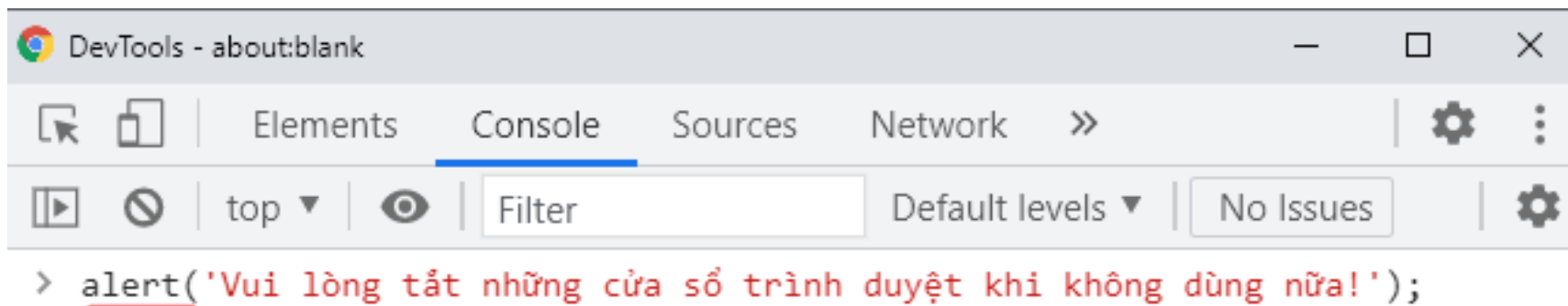
❑ Lệnh **alert(..)**:

➤ Phương thức **alert(..)** là phương thức **hỗ trợ hiển thị một hộp thoại cảnh báo với một tin nhắn và một nút OK** ra trình duyệt hiện tại.

➤ Cú pháp: **alert(*message*)**

▪ *message*: Là đoạn tin nhắn hoặc đối tượng được chuyển đổi từ một chuỗi được hiển thị trong 1 hộp thoại trên trình duyệt.

➤ Ví dụ: hiển thị thông báo với đoạn chữ “**Vui lòng tắt những cửa sổ trình duyệt khi không dùng nữa!**” ra màn hình trình duyệt.

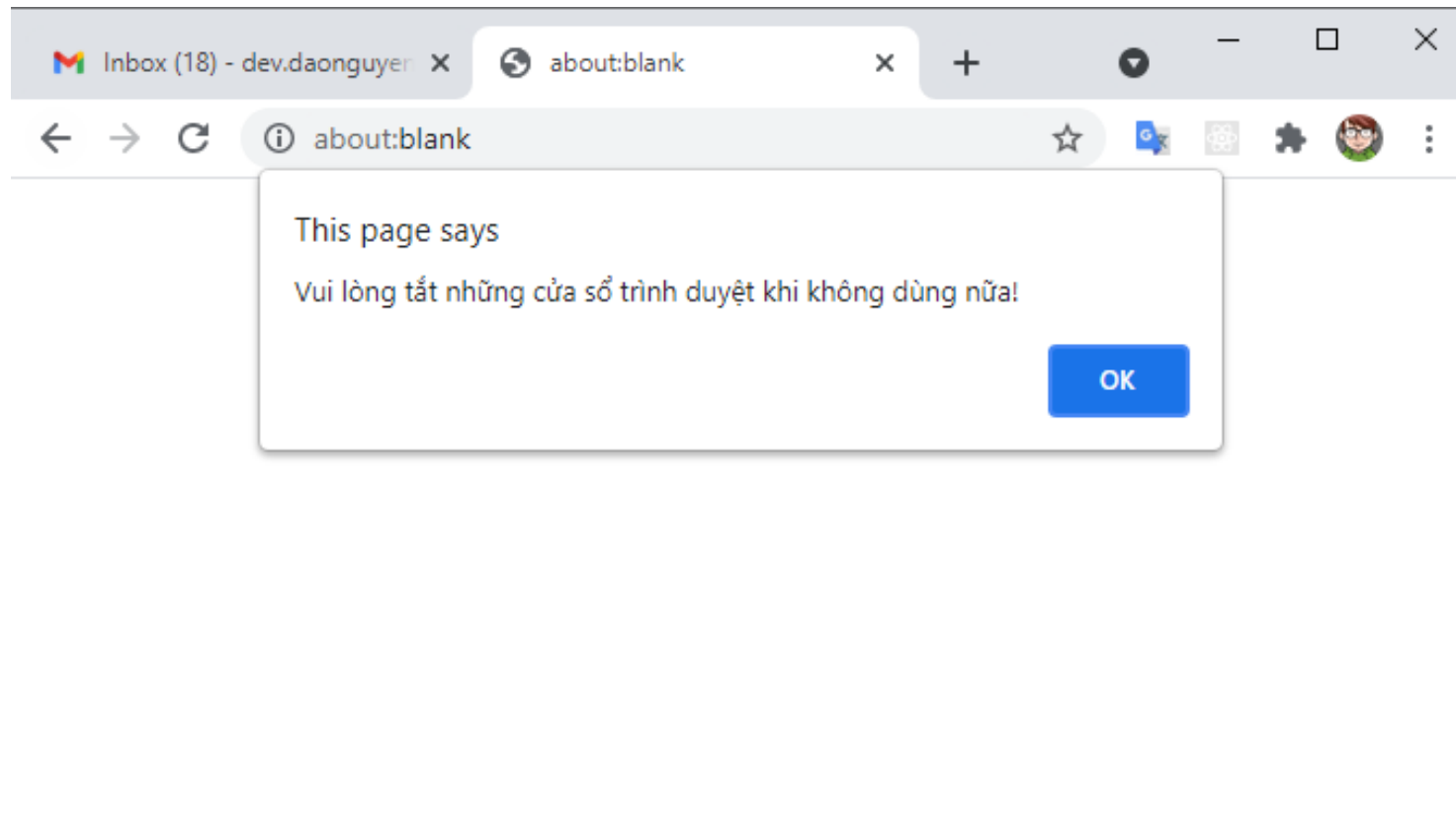




Các lệnh cơ bản (statements)

❑ Lệnh **alert(..)**:

➤ Ví dụ: hiển thị thông báo với đoạn chữ “**Vui lòng tắt những cửa sổ trình duyệt khi không dùng nữa!**” ra màn hình trình duyệt.





Các lệnh cơ bản (statements)

❑ Lệnh **prompt(..)**:

- Phương thức **prompt(..)** là phương thức **hỗ trợ hiển thị một hộp thoại cho phép người dùng nhập dữ liệu vào ô nhập liệu.**
- Khi người dùng nhập liệu xong, người dùng có 2 nút “OK” và “Cancel” mà người dùng có thể nhấn:
 - Nhấn “**Cancel**”: **giá trị trả về** cho phương thức **prompt** này sẽ là **null**.
 - Nhấn “**OK**”: **giá trị trả về** cho phương thức **prompt** này là **dữ liệu người dùng đã nhập.**



Các lệnh cơ bản (statements)

❑ Lệnh **prompt(..)**:

➤ Cú pháp: **prompt**(*text*, *defaultText*)

- *text*: Là **đoạn một chuỗi** được **hiển thị trong 1 hộp thoại** trên trình duyệt (**Bắt buộc có**).

- *defaultText*: Là **giá trị mặc định** cho ô nhập (**Không bắt buộc có**).

- Ví dụ: hiển thị câu hỏi sau “**Cho tui xin facebook của bạn đi?**” ra màn hình trình duyệt của một bạn nữ với đoạn mã sau.

```
> var question = "Cho tui xin facebook của bạn đi?";
   var answer = prompt(question);

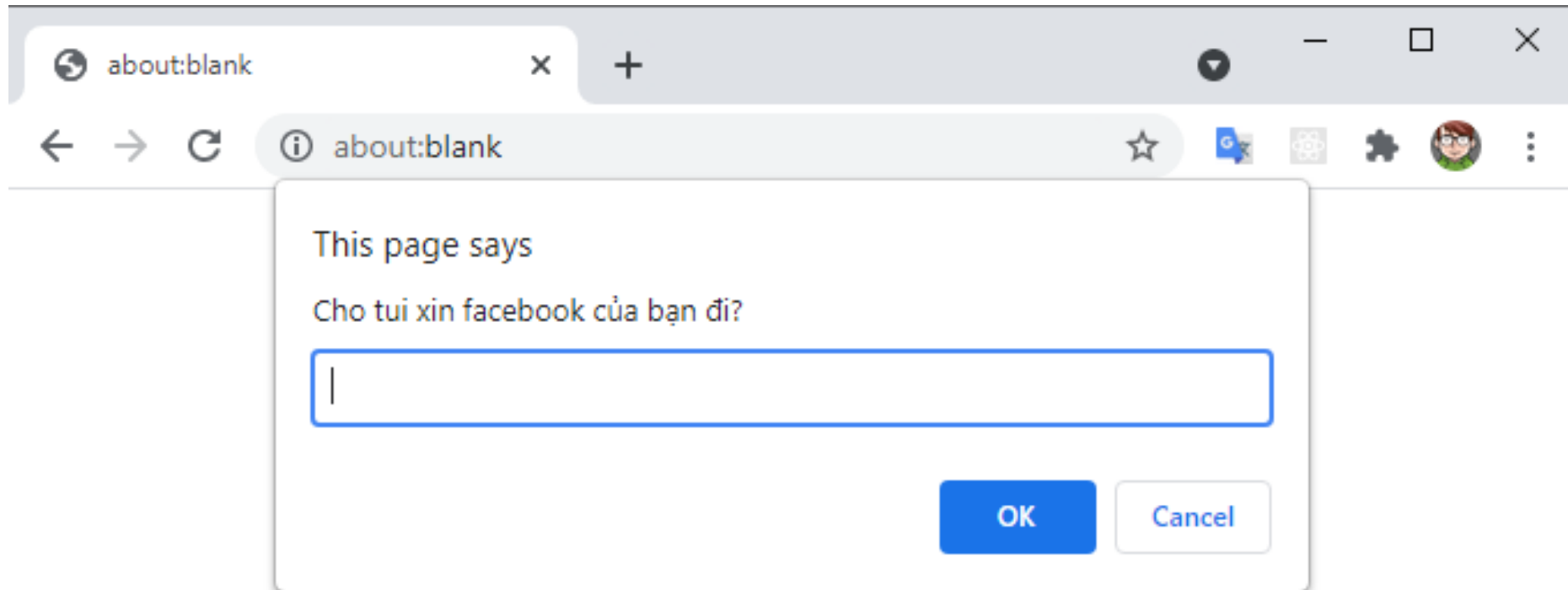
   if (answer != null) {
       console.log('Nick tui nè: ' + answer);
   } else {
       console.log('Nhìn mặt thì ghét, không cho!');
   }
```



Các lệnh cơ bản (statements)

❑ Lệnh **prompt(..)**:

➤ Ví dụ: hiển thị câu hỏi sau “**Cho tui xin facebook của bạn đi?**” ra màn hình trình duyệt. Dùng if-else để bắt giá trị trả về của **prompt** và hiển thị một câu trả lời tùy ý. (if-else sẽ học sau)

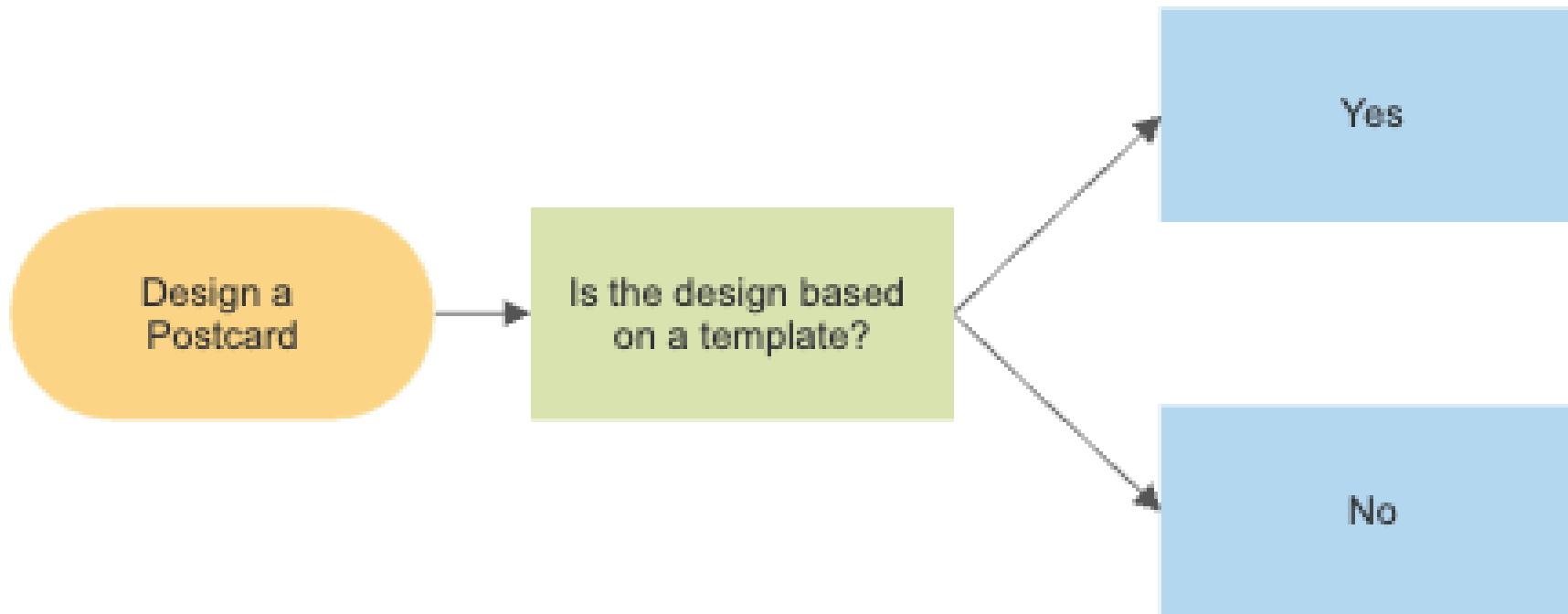




Lưu đồ thuật toán (flowchart)

□ Định nghĩa:

➤ Lưu đồ thuật toán (flowchart) là **công cụ dùng để biểu diễn thuật toán, mô tả dữ liệu nhập (input), dữ liệu xuất (output) và luồng xử lý thuật toán** thông qua các ký hiệu hình học.

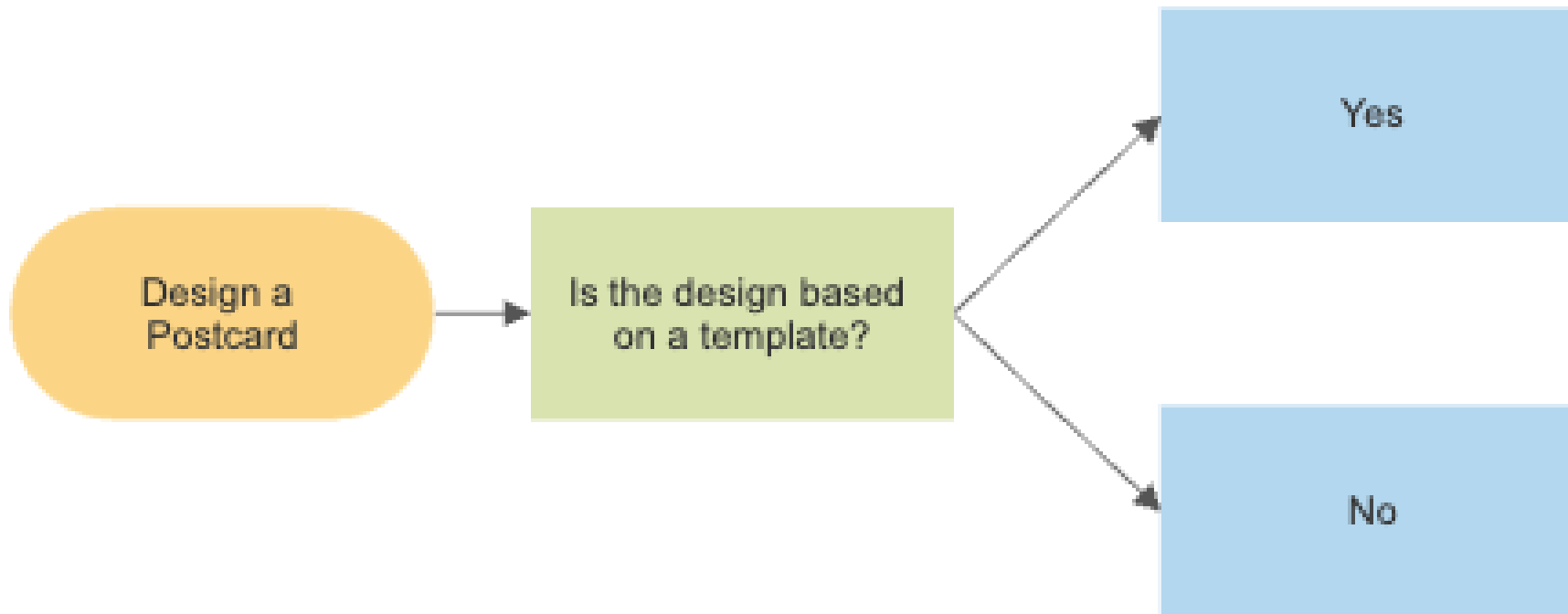




Lưu đồ thuật toán (flowchart)

□ Công dụng:

➤ **Giúp kiểm tra tính khả thi** của **việc lập trình**, **đưa ra** những **giải thuật** để **viết chương trình** một cách **nhANH chóng**, **hiệu quả** qua các bước trong lưu đồ được thực hiện theo một trình tự đơn giản.





Lưu đồ thuật toán (flowchart)

□ Các ký hiệu:

➤ Để vẽ lưu đồ thuật toán, bạn cần nhớ và **tuân thủ các ký hiệu sau**:

STT	Ký hiệu	Ý nghĩa
1		Bắt đầu / Kết thúc chương trình
2		Điều kiện rẽ nhánh (lựa chọn)
3		Luồng xử lý
4		Nhập



Lưu đồ thuật toán (flowchart)

□ Các ký hiệu:

➤ Để vẽ lưu đồ thuật toán, bạn cần nhớ và **tuân thủ các ký hiệu sau**:

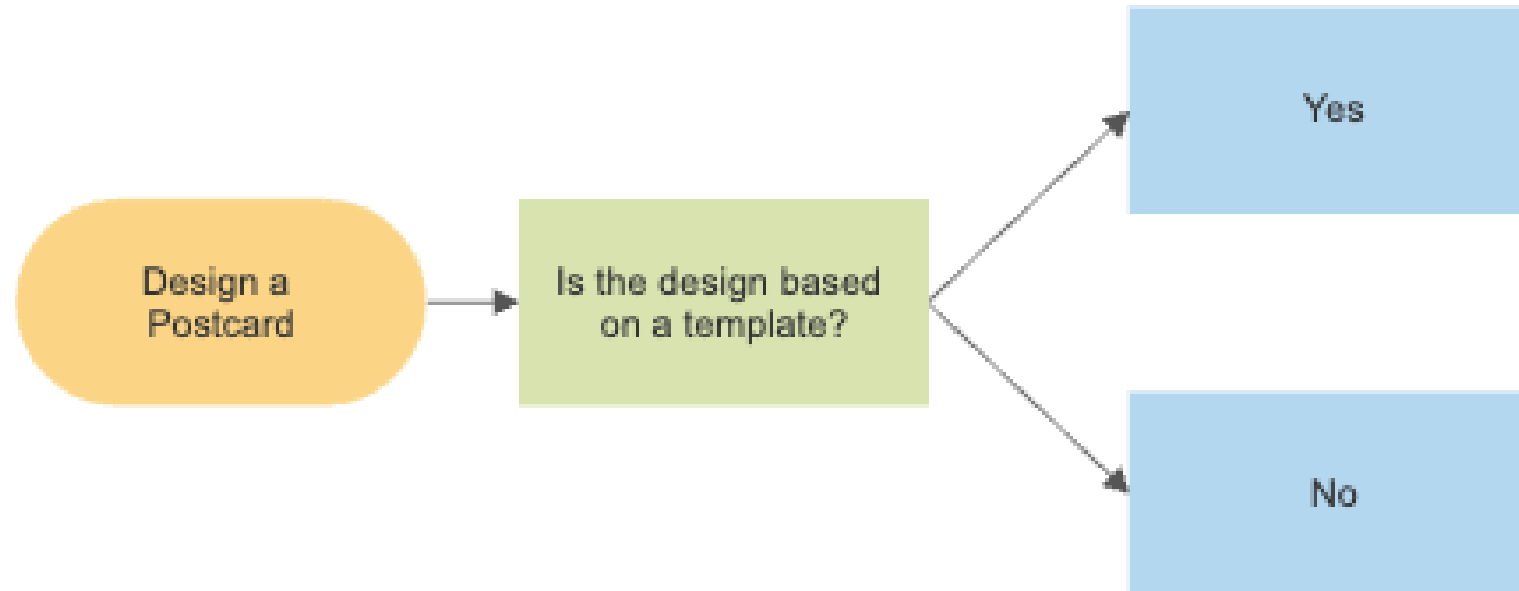
5		Xuất
6		Xử lý / tính toán / gán
7		Trả về giá trị (return)
8		Điểm nối liên kết



Lưu đồ thuật toán (flowchart)

□ Trình tự:

- Lưu đồ thuật toán được duyệt theo trình tự:
 - Duyệt từ **trên xuống dưới**.
 - Duyệt từ **trái sang phải**.

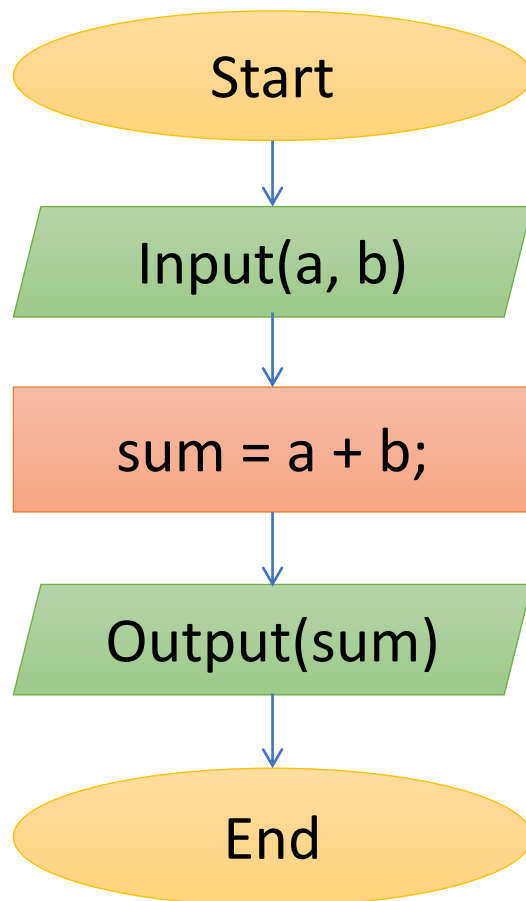




Lưu đồ thuật toán (flowchart)

❑ Ví dụ:

- Vẽ lưu đồ thuật toán nhập 2 số a và b. Tính tổng 2 số và xuất ra kết quả.



```
DevTools - about:blank
```

Elements Console Sources Network

top Filter Default

```
> var a = prompt("Mời bạn nhập số hạng a: ");  
var b = prompt("Mời bạn nhập số hạng b: ");  
var sum = Number(a) + Number(b);  
console.log("Tổng của 2 số a và b là: " + sum);
```

Tổng của 2 số a và b là: 11



Tổng kết:

- ☐ Biến (variables) & hằng (constants)
- ☐ Giá trị (values) & kiểu (types)
- ☐ Biểu thức (expressions) & toán tử (operators)
- ☐ Khối lệnh (blocks)
- ☐ Các lệnh cơ bản (statements)
- ☐ Lưu đồ thuật toán (flowchart)

Let's
Recap

