# JavaScript learning path

Perfect timing 🚀 — let's now move into **Conditional Branching in JavaScript** (the real decision-making backbone).
We'll go step by step, from surface usage → internal behavior → pitfalls → interview-level traps.

---

## 🧩 Conditional Branching in JavaScript

---

### 1. What It Means

Conditional branching = **executing different code paths depending on conditions** (true/false decisions).
JS supports several branching constructs:

1. `if / else`
2. `switch`
3. Ternary (`condition ? expr1 : expr2`)
4. (indirect branching via short-circuit `&&` / `||`)

---

### 2. `if / else` (the classic)

```js
let age = 20;

if (age >= 18) {
  console.log("Adult");
} else {
  console.log("Minor");
}
```

#### Internal behavior:

- Expression inside `if()` is evaluated.
- The result is **coerced to boolean** using `ToBoolean` rules:
  - Falsy: `false, 0, -0, 0n, "", null, undefined, NaN`

- Everything else → truthy (even `"0"`, `"false"`, `[]`, `{}`)

👉 This explains why:

```js
if ("0") console.log("runs"); // "runs" (non-empty string → truthy)
if ([]) console.log("runs");  // "runs" (empty array → truthy)
```

---

## 3. `else if`

```js
let score = 85;

if (score >= 90) {
  console.log("A");
} else if (score >= 75) {
  console.log("B");
} else {
  console.log("C");
}
```

Internally:

- Evaluates sequentially top to bottom.
- First branch that resolves to `true` executes.
- **No backtracking** (it doesn't check others once matched).

---

## 4. Ternary Operator `? :`

```js
let age = 20;
let status = (age >= 18) ? "Adult" : "Minor";
console.log(status); // Adult
```

Internally:

- Compact form of `if / else`.
- Returns a value (unlike `if` which executes statements).

- **Right-associative:** `a ? b : c ? d : e` → parsed as `a ? b : (c ? d : e)`

---

## 5. `switch`

```js
let color = "red";

switch (color) {
  case "red":
    console.log("Stop");
    break;
  case "green":
    console.log("Go");
    break;
  default:
    console.log("Wait");
}
```

### Internal behavior:

- Uses **strict equality (`===`)** for comparisons.
- Evaluates top to bottom until match.
- Without `break`, execution “falls through” to next case.

Example pitfall:

```js
let x = "1";

switch (x) {
  case 1: console.log("number one"); break;
  case "1": console.log("string one"); break;
}
// output → "string one" (strict equality)
```

---

## 6. Short-Circuit Branching (`&&` and `||`)

JS allows boolean-like control flow with logical operators:

```js
let loggedIn = true;
let username = loggedIn && "Alice";

console.log(username); // "Alice"
```

- `&&` returns **first falsy** or last value.
- `||` returns **first truthy** or last value.

```js
console.log(null || "default"); // "default"
console.log("value" && "next"); // "next"
```

👉 This is why these are often used as **inline conditionals**.

---

## 7. Pitfalls

1. **Falsy traps**

   ```js
   if (0) console.log("runs"); // won't run
   if ("0") console.log("runs"); // runs
   ```

2. **switch fall-through**

   ```js
   let n = 1;
   switch (n) {
     case 1: console.log("one");
     case 2: console.log("two");
   }
   // one
   // two
   ```

3. **Assignment vs comparison**

   ```js
   ```

```
let a = 5;
if (a = 0) console.log("oops"); // assignment! always falsy
```

4. **Nested ternary readability**

```js
let msg = isAdmin ? "Admin" : isUser ? "User" : "Guest";
// Confusing to debug
```

---

## 8. Interview Traps

- Q: Difference between `==` and `===` in `switch`?

  👉 `switch` always uses `===`.
- Q: Why does `if([])` execute?

  👉 Because empty array is an object → always truthy.
- Q: How to avoid switch fall-through?

  👉 Use `break` or `return` inside cases.
- Q: Can you replace `if/else` with ternary everywhere?

  👉 Not always; ternary is an *expression*, not a *statement*.

---

### ✅ Summary

- Branching is built on `ToBoolean` coercion.
- `if/else` → sequential truthy/falsy checks.
- `switch` → strict equality chain.
- `ternary` → compact conditional expression.
- `&&` / `||` → short-circuit branching (truthy/falsy return).

---

⚡ Next step: I can either

1. Deep dive into **truthy/falsy coercion rules** (internal conversion table), OR
2. Move on to **Loops (iteration)**.

👉 Which one would you like to explore first?