JavaScript learning path

Perfect ✅ — we'll **stay within what we've already covered:** `while`, `do...while`, `for`, `break`, `continue`, and labels.

I'll now give you a **deep dive** into these loops, with:

- Internal behavior
- Common pitfalls
- Brain-vender / interview-style tricky questions

🔁 Deep Dive into Loops (while, do...while, for)

---

1. `while` loop

✅ Syntax

```js
while (condition) {
  // loop body
}
```

🔎 Internal Behavior

1. Evaluate `condition` → convert to boolean (`truthy/falsy`).
2. If `true` → execute body → repeat.
3. If `false` → stop immediately.
4. Condition is checked **before** body every time.

⚠ Pitfalls

- **Infinite loop** if you forget to update variables:

```js
let i = 0;
while (i < 5) {
  console.log(i); // runs forever
}
```

- **Condition shortcut**:
  `while (i)` is same as `while (i != 0)` because non-zero = truthy.

---

2. `do…while` loop

✅ Syntax

```js
do {
  // loop body
} while (condition);
```

🔎 Internal Behavior

1. Body runs **once without condition check**.
2. Then condition is checked.
3. If `true` → repeat body.
4. If `false` → stop.

⚠ Pitfalls

- Runs at least once → can surprise beginners:

  ```js
  let i = 10;
  do {
    console.log(i); // prints 10
  } while (i < 5); // still runs once
  ```

---

3. `for` loop

✅ Syntax

```js
```

```
for (begin; condition; step) {
  // loop body
}
```

🔎 Internal Behavior

1. Run `begin` once.
2. Check `condition`. If false → stop.
3. Run body.
4. Run `step`.
5. Repeat condition → body → step until false.

⚠ Pitfalls

- Forgetting semicolons → syntax error:

  ```js
  for (let i = 0 i < 5 i++) {} // ❌ wrong
  ```

- Missing parts still valid:

  ```js
  for (;;) { } // infinite loop
  ```

- Scope confusion:

  ```js
  for (let i = 0; i < 3; i++) {}
  console.log(i); // ❌ ReferenceError (i is block-scoped)
  ```

---

4. `break` & `continue`

✅ Break

Stops loop immediately.

```js
for (let i = 0; i < 5; i++) {
  if (i === 3) break;
```

```
  console.log(i); // 0,1,2
}
```

✅ Continue

Skips to next iteration.

```js
for (let i = 0; i < 5; i++) {
  if (i % 2 === 0) continue;
  console.log(i); // 1,3
}
```

⚠ Pitfalls

- Continue only skips current iteration, doesn’t stop loop.
- `break` only exits the **closest loop**.

---

5. Labels

✅ Syntax

```js
labelName: for (...) {
  ...
}
```

Example

```js
outer: for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {
    if (i === 1 && j === 1) break outer;
    console.log(i, j);
  }
}
```

👉 Breaks out of both loops when condition matches.

⚠ Pitfalls

- Labels **cannot jump anywhere**. Only valid with `break` / `continue`.
- Misuse leads to unreadable "spaghetti code".

🧠 Brain-Vender Interview Questions on Loops

Q1. What will this print?

```js
let i = 0;
while (i < 5) {
  console.log(i++);
}
```

Answer: `0 1 2 3 4`

👉 Post-increment → prints first, then increases.

Q2. Infinite loop trap

```js
for (let i = 0; i != 10; i += 0.2) {
  // do something
}
```

Answer: May never terminate.

👉 Floating-point arithmetic can skip exactly `10` (e.g., 9.8 → 10.0 never reached).

Q3. Scope trap

```js
for (var i = 0; i < 3; i++) {}
console.log(i); // ?
```

Answer: `3`

👉 `var` is function-scoped, survives outside loop.

With `let` → ReferenceError.

Q4. `do…while` trick

```js
let i = 0;
do {
  console.log(i);
} while (i > 0);
```

Answer: Prints `0` once.

👉 Condition false, but body runs once anyway.

Q5. Break confusion

```js
for (let i = 0; i < 3; i++) {
  for (let j = 0; j < 3; j++) {
    break;
  }
  console.log(i);
}
```

Answer: Prints `0 1 2`.

👉 `break` only exits inner loop, not outer.

✅ This completes a **deep dive into basic loops** (while, do…while, for, break/continue, labels).

Would you like me to now move to **real-world use cases of these loops (arrays, user input, simulation tasks)** OR keep exploring **more tricky pitfalls and interview problems** first?