JavaScript learning path

Perfect 🚀 Let’s move on to the next primitive:

🟣 `undefined` in JavaScript (Deep Dive)

---

1. 🔹 What is `undefined`?

- A special **primitive value** that represents:

  > *“a variable has been declared but not assigned any value yet.”*

- It is **not** the same as `null`.
- Type:

  ```js
  typeof undefined; // "undefined"
  ```

---

2. 🔹 When do you get `undefined`?

(a) Variable declared but not assigned

```js
let x;
console.log(x); // undefined
```

(b) Missing function return

```js
function test() {}
console.log(test()); // undefined
```

(c) Accessing missing object properties

```js
let obj = {};
```

```
console.log(obj.key); // undefined
```

(d) Array element not set

```js
let arr = [1, , 3]; // sparse array
console.log(arr[1]); // undefined
```

(e) Function parameters not passed

```js
function greet(name) {
  console.log(name);
}
greet(); // undefined
```

---

3. 🔹 Difference between `undefined` and `null`

Feature	`undefined`	`null`
Meaning	Declared but not assigned, missing value	Explicitly no value
Type	`"undefined"`	`"object"` (quirk of JS)
Set by	JavaScript engine	Developer (intentionally)
Example	`let a; console.log(a); // undefined`	`let a = null;`

---

4. 🔹 Type Coercion with `undefined`

```js
```

```
Number(undefined);   // NaN
String(undefined);   // "undefined"
Boolean(undefined);  // false
```

Loose equality traps:

```js
undefined == null;  // true  (special case)
undefined === null; // false (different types)
```

---

5. 🔹 Edge Cases

1. **Global variable `undefined` is writable in old JS**

   ```js
   undefined = "oops";
   console.log(undefined); // "oops" (in ES3/ES5 non-strict mode)
   ```

   ✅ In modern JS (ES5 strict mode + ES6+), `undefined` is read-only.

2. **Array holes vs `undefined`**

   ```js
   let arr = [1, , 3];
   console.log(arr[1]);   // undefined
   console.log(1 in arr); // false (hole, not assigned `undefined`)
   ```

3. **JSON.stringify drops `undefined`**

   ```js
   JSON.stringify({a: undefined, b: 2});
   // "{"b":2}"
   ```

---

6. 🔹 Best Practices

- ✅ Use `===` when comparing with `undefined`.
- ✅ Prefer `null` when you want to **explicitly clear/reset a variable**.

- ❌ Don't manually assign `undefined`, use `null` instead.

  ```js
  let user = null; // clear value
  ```

🧩 Quick Exercise (Your Turn 😉)

What will be the output?

```js
let x;
console.log(x == null);   // ?
console.log(x === null);  // ?
console.log(x === undefined); // ?
```

👉 Do you want me to **solve this right now step by step**, or first give you a few more **brain-bender corner cases with `undefined`**?

Alright 💡 let's solve the exercise step by step, and then I'll also add some **brain-bender corner cases** around `undefined`.

🔎 Exercise Solution

```js
let x;
console.log(x == null);      // ?
console.log(x === null);     // ?
console.log(x === undefined);// ?
```

Step 1: `let x;`

- Variable `x` is declared but not assigned.
- So `x` → `undefined`.

Step 2: `x == null`

- **Loose equality (`==`)** does **type coercion**.
- Special rule:
  👉 `null == undefined` → **true**.
- So:

  ```js
  console.log(undefined == null); // true
  ```

✅ Output: `true`

Step 3: `x === null`

- **Strict equality (`===`)** checks type + value.
- `x` is `undefined`, not `null`.
- Types differ: `"undefined"` vs `"object"`.
- So:

  ```js
  console.log(undefined === null); // false
  ```

✅ Output: `false`

Step 4: `x === undefined`

- `x` is literally `undefined`.
- Types and values match.
- So:

  ```js
  console.log(undefined === undefined); // true
  ```

✅ Output: `true`

🎯 Final Answer

```js
true
false
true
```

🧩 Extra Brain-Benders (Interview-Style)

1. **Undefined in arrays**

```js
let arr = [undefined, ,];
console.log(arr.length); // ?
console.log(arr[0]);     // ?
console.log(arr[1]);     // ?
```

2. **JSON behavior**

```js
console.log(JSON.stringify({a: undefined, b: null}));
// ?
```

3. **Global undefined quirk**

```js
var undefined = 123;
console.log(undefined); // ?
```

👉 Do you want me to **solve these 3 undefined brain-benders** step by step right now, or should I move on to the next primitive: `null`?

Perfect 🙌 let’s solve these **3** `undefined` **brain-benders** step by step.

🧩 Brain-Bender 1: Undefined in arrays

```js
let arr = [undefined, ,];
console.log(arr.length); // ?
console.log(arr[0]);     // ?
console.log(arr[1]);     // ?
```

Step 1: Array creation

- `arr = [undefined, ,];` creates an array with **2 slots**:
  - `arr[0] = undefined` (explicitly set)
  - `arr[1] = <empty slot>` (called a *hole*)

Step 2: `console.log(arr.length);`

- Array has 2 indexes → length = **2**.

Step 3: `console.log(arr[0]);`

- Explicitly assigned `undefined`.
  ✅ Output → `undefined`

Step 4: `console.log(arr[1]);`

- This is **a hole**, not even `undefined` explicitly.
- When accessed, JS returns `undefined`, but internally `arr` doesn’t store a value there.

✅ Output → `undefined`

👉 Difference:

```js
console.log(0 in arr); // true  (index exists, holds undefined)
console.log(1 in arr); // false (hole, doesn’t exist)
```

🧩 Brain-Bender 2: JSON behavior

```js
console.log(JSON.stringify({a: undefined, b: null}));
```

Rule:

- **`JSON.stringify` drops properties with `undefined` values.**
- `null` is preserved.

So:

```js
{a: undefined, b: null} → {"b":null}
```

✅ Output → `{"b":null}`

🧩 Brain-Bender 3: Global undefined quirk

```js
var undefined = 123;
console.log(undefined);
```

In old JS (ES3, ES5 non-strict):

- `undefined` is just a **global variable.**
- It can be reassigned.
  ✅ Output → `123`

In modern JS (ES5 strict mode / ES6+):

- `undefined` is **read-only** and non-writable.
- Attempting to overwrite has **no effect.**
  ✅ Output → `undefined`

✅ Final Answers

1. Array trick:

```javascript
2
undefined
undefined
```

(but note: index `0` exists, index `1` is a hole)

2. JSON trick:

```json
{"b":null}
```

3. Global trick:

```javascript
123   // old non-strict
undefined // modern strict
```

⚡ Interview Tip:

If asked **“Is `undefined` a keyword in JS?”** → Answer:

No, it’s a **global property**. It only became effectively read-only from ES5 strict mode onward.

👉 Do you want me to **move to the next primitive: `null`** now, or would you like me to show `undefined` **in function parameter defaults and destructuring** first (another hidden pitfall)?