Below is a **clear, exam-ready + industry-correct explanation** of **Custom Errors in JavaScript**, structured from **concept → internal working → methods → examples → best practice**.

🛑 Custom Errors in JavaScript

1️⃣ What are Custom Errors?

Custom errors are **user-defined error types** created to represent **specific application-level problems** more clearly than generic errors.

Instead of throwing vague errors like:

```
js

Error: Something went wrong
```

we can throw **meaningful errors** like:

```
js

AgeNotValidError: Age must be 18 or above
```

📌 This improves:

- Readability
- Debugging
- Error categorization
- User-friendly error handling

2️⃣ Why Do We Need Custom Errors?

Built-in errors (`Error`, `TypeError`, `RangeError`, etc.) are **generic**.

In real applications, we need **domain-specific errors**, for example:

- `AgeNotValidError`
- `AuthenticationError`
- `DatabaseConnectionError`
- `HttpError`
- `ValidationError`

👉 Custom errors help distinguish **what failed** and **why**.

3️⃣ JavaScript Error Class (Internal Understanding)

Whenever an error occurs, JavaScript throws an **Error object**.

Internally, Error looks like:

```
js

class Error {
  constructor(message) {
    this.message = message;
    this.name = "Error";
    this.stack = <call stack>;
  }
}
```

Important properties:

Property	Meaning
`name`	Type of error
`message`	Description
`stack`	Execution trace (non-standard)

4️⃣ Method 1: Custom Error Using Error Instance (Simplest)

Syntax

```
js

const customError = new Error("message");
customError.name = "CustomErrorName";
```

Example

```
html

<script>
  const customError = new Error("This is a custom error");
  customError.name = "CustomError";

  try {
    throw customError;
  } catch (err) {
    document.body.innerHTML = err;
```

```
}
</script>
```

Output

```
vbnet

CustomError: This is a custom error
```

✔️ Easy

❌ Not scalable for large apps

5️⃣ Method 2: Custom Error Using Function Constructor (Old Style)

This approach uses **prototype inheritance** (pre-ES6).

Syntax

```
js

function ValidationError(message, name) {
 this.message = message;
 this.name = name;
}
ValidationError.prototype = Error.prototype;
```

Example

```
html

<script>
 function ValidationError(message, name) {
  this.message = message;
  this.name = name;
 }
 ValidationError.prototype = Error.prototype;

 try {
  let value = 10;
  if (typeof value !== "string") {
   throw new ValidationError("Not a string", "NotStringError");
  }
 } catch (e) {
  document.body.innerHTML = e.name + ": " + e.message;
```

```
}
</script>
```

Output

```
vbnet

NotStringError: Not a string
```

✔️ Works in old JS

❌ Not recommended now

6️⃣ Method 3: Custom Error by Extending Error Class (BEST PRACTICE ✅)

This is the **modern and recommended approach**.

Syntax

```
js

class CustomError extends Error {
 constructor(message) {
  super(message);
  this.name = "CustomError";
 }
}
```

Example: Age Validation Error

```
html

<script>
 class AgeNotValidError extends Error {
  constructor(message) {
   super(message);
   this.name = "AgeNotValidError";
  }
 }

 try {
  let age = 16;
  if (age < 18) {
   throw new AgeNotValidError("Age must be 18 or above");
  }
```

```
  } catch (e) {
    document.body.innerHTML = e.name + ": " + e.message;
  }
</script>
```

Output

```
makefile

AgeNotValidError: Age must be 18 or above
```

✔️ Clean
✔️ Scalable
✔️ Works with `instanceof`
✔️ Industry standard

7️⃣ Using `instanceof` with Custom Errors

```
js

if (error instanceof AgeNotValidError) {
  // handle age issue
}
```

📌 This is only possible when using **class-based custom errors**.

8️⃣ When to Use Custom Errors?

Use custom errors when:

- Validating user input
- Handling API responses
- Enforcing business rules
- Separating error logic
- Building large applications

9️⃣ Comparison of All Approaches

Method	Recommended	Reason
Error instance	❌	Limited
Function constructor	❌	Legacy
`class extends Error`	✅	Modern, scalable

🔑 Final Summary

- Custom errors provide **meaningful error handling**
- JavaScript allows custom errors using:
  1. Error instances
  2. Function constructors
  3. Class inheritance (**best**)
- `extends Error` is the **industry standard**
- Custom errors improve **debugging, readability, and control**

If you want next, I can cover:

- 🔁 **Custom Errors with Promises**
- 🌐 **HTTP Error Handling**
- 🧪 **Interview Questions on Errors**
- 🧠 **Error Design Patterns**

Just tell me 👍