Strict Mode in JS:

Strict mode is a feature in JavaScript that enables stricter parsing and error handling of the code. When strict mode is enabled, certain behaviors that were previously allowed are now considered to be errors.

Strict mode can be applied to an entire script or to individual functions. To enable strict mode for an entire script, the following line of code should be placed at the top of the script:

```
"use strict";
```

To enable strict mode for an individual function, the following line of code should be placed at the top of the function:

```
function example() {
  "use strict";
  // function code here
}
```

When strict mode is enabled, the following behaviors are considered to be errors:

- Using a variable that has not been declared with the var, let, or const keyword.
- Deleting a variable, function, or a property of an object.
- Using a reserved word as a variable or function name.
- Using with statement
- Duplicating a parameter name in a function.
- Modifying a read-only property.
- Octal numeric literals (e.g. 08)
- Strict mode also makes some optimizations possible, such as faster property access, and provides better error messages, which makes it easier to find and fix errors in the code.

It's worth noting that strict mode is not mandatory and it's up to the developer to decide when to use it. Enabling strict mode in parts of the codebase can help to catch errors early in development and make the codebase more maintainable.

Try-Catch-Finally Block in JS:

In JavaScript, the try-catch block is used to handle errors that may occur in a piece of code. The try block contains the code that may throw an error, and the catch block contains the code that will handle the error.

For example:

```
try {
  let x = y; // ReferenceError: y is not defined
} catch (error) {
  console.log(error.message); // Output: y is not defined
}
```

In this example, the try block tries to assign the value of a variable y to a variable x, but y has not been defined. This causes a ReferenceError to be thrown, which is caught by the catch block. The error object is passed to the catch block as an argument and can be used to access information about the error, such as the error message.

The try-catch block can also be used with the finally block, which contains code that will always be executed regardless of whether an error was thrown or not.

```
try {
  let x = y;
} catch (error) {
  console.log(error.message);
} finally {
  console.log("This code will always run");
}
```

The try-catch block is a useful tool for handling errors in JavaScript and can help to prevent the code from crashing when an error occurs. It also makes it possible to handle different types of errors in different ways, and to provide a more user-friendly experience for the end-user.

Custom Errors in JS:

In JavaScript, it's possible to create custom error classes by extending the built-in Error class. This allows you to create specific error types for your application, making it easier to handle and diagnose errors.

For example, you can create a custom error class called ValidationError:

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

This custom error class can be used to throw an error when validation fails:

```
function validate(value) {
  if (value < 0) {
    throw new ValidationError("Value must be greater than 0");
  }
}

try {
  validate(-1);
} catch (error) {
  if (error.name === "ValidationError") {
    console.log(error.message); // Output: "Value must be
greater than 0"
  } else {
    throw error;
  }
}
```

Custom error classes can be useful for providing more context and information about an error, as well as making it easier to identify and handle specific types of errors. They

can also be used to create specific error types for different layers of your application, such as validation errors, network errors, and database errors.

It's worth noting that custom errors should have a name property and it's highly recommended to have a message property, for the ease of debugging and handling the error in the catch block.