# Basic Error Handling in JavaScript

## 1. Try...Catch Statement

The try...catch statement lets you test a block of code for errors (try) and handle them (catch).

### Syntax:

```
try {
  // Block of code to try
} catch (error) {
  // Block of code to handle errors
}
```

### Example:

```
try {
  let result = someFunction();
  console.log(result);
} catch (error) {
  console.error('An error occurred:', error.message);
}
```

## 2. Finally Clause

The finally clause can be added after try and catch blocks. It executes code regardless of whether an error was thrown or caught.

### Syntax:

```
try {
  // Block of code to try
} catch (error) {
  // Block of code to handle errors
} finally {
  // Block of code to be executed regardless of try/catch result
}
```

### Example:

```
try {
  let result = someFunction();
  console.log(result);
} catch (error) {
  console.error('An error occurred:', error.message);
} finally {
  console.log('This will always execute');
}
```

## 3. Throw Statement

The throw statement allows you to create custom errors. You can use it to throw exceptions.

### Syntax:

```
throw new Error('Something went wrong');
```

### Example:

```
function checkAge(age) {
  if (age < 18) {
    throw new Error('You must be at least 18 years old');
  }
  return 'Access granted';
}

try {
  let message = checkAge(16);
  console.log(message);
} catch (error) {
  console.error('An error occurred:', error.message);
}
```

## 4. Types of Errors

JavaScript has several built-in error types you can use with throw:

- Error: A generic error.
- SyntaxError: An error in the syntax.
- ReferenceError: An illegal reference.
- TypeError: A type mismatch.
- RangeError: A number out of range.
- URIError: An error in URI handling.

### Example:

```
try {
  throw new TypeError('This is a type error');
} catch (error) {
  console.error(error.name + ': ' + error.message);
}
```

## 5. Custom Error Handling

Creating custom error classes allows you to define more specific error types.

### Example:

```
class CustomError extends Error {
  constructor(message) {
    super(message);
```

```
    this.name = 'CustomError';
  }
}

try {
  throw new CustomError('This is a custom error');
} catch (error) {
  console.error(error.name + ': ' + error.message);
}
```

## Best Practices

1. Use Specific Error Types: Differentiate between different types of errors using specific error classes.
2. Provide Useful Error Messages: Ensure error messages are informative.
3. Don’t Swallow Errors: Always handle or log errors to avoid silent failures.
4. Cleanup in finally: Use the finally block to release resources or perform cleanup tasks.