JavaScript try...catch...finally Statement

The `try`, `catch` and `finally` blocks are used to handle exceptions (a type of an error). Before you learn about them, you need to know about the types of errors in programming.

Types of Errors

In programming, there can be two types of errors in the code:

Syntax Error: Error in the syntax. For example, if you write `consol.log('your result');`, the above program throws a syntax error. The spelling of `console` is a mistake in the above code.
Runtime Error: This type of error occurs during the execution of the program. For example,
calling an invalid function or a variable.

These errors that occur during runtime are called **exceptions**. Now, let's see how you can handle these exceptions.

JavaScript try...catch Statement

The `try...catch` statement is used to handle the exceptions. Its syntax is:

```
try {
    // body of try
}
catch(error) {
    // body of catch
}
```

The main code is inside the `try` block. While executing the `try` block, if any error occurs, it goes to the `catch` block. The `catch` block handles the errors as per the catch statements.

If no error occurs, the code inside the `try` block is executed and the `catch` block is skipped.

Example 1: Display Undeclared Variable

```
// program to show try...catch in a program

const numerator= 100, denominator = 'a';

try {
     console.log(numerator/denominator);

    // forgot to define variable a
    console.log(a);
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}
```

Run Code

Output

```
NaN
An error caught
Error message: ReferenceError: a is not defined
```

In the above program, `a` variable is not defined. When you try to print the `a` variable, the program throws an error. That error is caught in the `catch` block.

JavaScript try...catch...finally Statement

You can also use the `try...catch...finally` statement to handle exceptions. The `finally` block executes both when the code runs successfully or if an error occurs.

The syntax of `try...catch...finally` block is:

```
try {
    // try_statements
}
catch(error) {
    // catch_statements
}
finally() {
    // codes that gets executed anyway
}
```

Example 2: try...catch...finally Example

```
const numerator= 100, denominator = 'a';

try {
     console.log(numerator/denominator);
     console.log(a);
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}
finally {
     console.log('Finally will execute every time');
}
```

Run Code

Output

```
NaN
An error caught
Error message: ReferenceError: a is not defined
Finally will execute every time
```

In the above program, an error occurs and that error is caught by the `catch` block. The `finally` block will execute in any situation (if the program runs successfully or if an error occurs).

Note: You need to use `catch` or `finally` statement after `try` statement. Otherwise, the program will throw an error `Uncaught SyntaxError: Missing catch or finally after try`.

JavaScript try...catch in setTimeout

The `try...catch` won't catch the exception if it happened in "**timed**" code, like in setTimeout(). For example,

```
try {
    setTimeout(function() {
        // error in the code
    }, 3000);
} catch (e) {
  console.log( "won't work" );
}
```

The above `try...catch` won't work because the engine has already left the `try..catch` construct and the function is executed later.

The `try..catch` block must be inside that function to catch an exception inside a timed function. For example,

```
setTimeout(function() {
    try {
        // error in the code
    } catch {
        console.log( "error is caught" );
    }
}, 3000);
```

You can also use the `throw` statement with the `try...catch` statement to use user-defined exceptions. For example, a certain number is divided by **0**. If you want to consider `Infinity` as an error in the program, then you can

throw a user-defined exception using the `throw` statement to handle that condition.

JavaScript throw Statement

In the previous tutorial, you learned to handle exceptions using JavaScript try..catch statement. The try and catch statements handle exceptions in a standard way which is provided by JavaScript. However, you can use the `throw` statement to pass user-defined exceptions.

In JavaScript, the `throw` statement handles user-defined exceptions. For example, if a certain number is divided by **0**, and if you need to consider `Infinity` as an exception, you can use the `throw` statement to handle that exception.

JavaScript throw statement

The syntax of throw statement is:

```
throw expression;
```

Here, `expression` specifies the value of the exception.

For example,

```
const number = 5;
throw number/0; // generate an exception when divided by 0
```

Note: The expression can be string, boolean, number, or object value.

JavaScript throw with try...catch

The syntax of `try...catch...throw` is:

```
try {
    // body of try
    throw exception;
}
catch(error) {
    // body of catch
}
```

Note: When the throw statement is executed, it exits out of the block and goes to the `catch` block. And the code below the `throw` statement is not executed.

Example 1: try...catch...throw Example

```
const number = 40;
try {
    if(number > 50) {
        console.log('Success');
    }
    else {

        // user-defined throw statement
        throw new Error('The number is low');
    }

    // if throw executes, the below code does not execute
    console.log('hello');
}
catch(error) {
    console.log('An error caught');
    console.log('Error message: ' + error);
}
```

Run Code

Output

```
An error caught
Error message: Error: The number is low
```

In the above program, a condition is checked. If the number is less than **51**, an error is thrown. And that error is thrown using the `throw` statement.

The `throw` statement specifies the string `The number is low` as an expression.

Note: You can also use other built-in error constructors for standard errors: `TypeError`, `SyntaxError`, `ReferenceError`, `EvalError`, `InternalError`, and `RangeError`.

For example,

```
throw new ReferenceError('this is reference error');
```

Rethrow an Exception

You can also use `throw` statement inside the `catch` block to rethrow an exception. For example,

```
const number = 5;
try {
     // user-defined throw statement
     throw new Error('This is the throw');

}
catch(error) {
    console.log('An error caught');
    if( number + 8 > 10) {

        // statements to handle exceptions
         console.log('Error message: ' + error);
        console.log('Error resolved');
    }
    else {
        // cannot handle the exception
        // rethrow the exception
        throw new Error('The value is low');
    }
}
```

Run Code

Output

```
An error caught
Error message: Error: This is the throw
Error resolved
```

In the above program, the `throw` statement is used within the `try` block to catch an exception. And the `throw` statement is rethrown in the `catch` block which gets executed if the `catch` block cannot handle the exception.
Here, the `catch` block handles the exception and no error occurs. Hence, the `throw` statement is not rethrown.
If the error was not handled by the catch block, the throw statement would be rethrown with error message `Uncaught Error: The value is low`