Try...catch

Syntax

```
try {
  // Code
}
catch (err) {
  // Error handling
}
finally {
   // Excuted always, regardless if there is any error. And is always last
    // It is also executed right before you return if you decide to return/throw from try and
catch
}
```

Keep in mind that `try...catch` works synchronously, so if you have asynchronous code and it throws an error after the execution of the `try...catch` block, then the error will not be caught.

In order to catch some exception in an asynchronous code, the `try...catch` must be inside the asynchronous function.

Error object

After an error has occurred inside the `try...catch` object, JavaScript generates an object containing the details about it

- `name`: Name of the error that has occurred
- `message`: Textual message about error details
- `stack`: The current call stack

By default, if you print the `err` object, it will print all three of these information, `name, message, and stack`.

Throwing our own errors

We can use the `throw` operator to generate our own error.

Technically we can throw anything as the error object, like primitives, numbers or strings, but is better if you throw an object.

```
let error = new Error(message);

try {
  throw error;
}
catch (err) {
  console.log(err);
}
```

There are many built-in constructors for standard errors: `Error`, `SyntaxError`, `ReferenceError`

Writing custom errors

To write your own custom error classes you can extend the `Error` with your own class

```
class CustomError extends Error {
  constructor(message) {
      super(message); // must be first line
        this.name = "CustomError";
      }
}

function test() {
  throw new CustomError("Custom test error thrown");
}

try {
    test();
}
catch (err) {
    console.log(err); // CustomError: Custom test error thrown + stack trace.
}
```

You can further extend your `CustomError` to be more specific errors and you can check if those extended errors are part of the `CustomError` by doing `if (err instanceof CustomError)`, if it is an error extended from `CustomError` then the result will be `true` otherwise, `false`.

Wrapping exceptions

Lower level exceptions can be re-raised as a more general exception under one category of exception and the real cause of the exception will be stored in it as a property `err.cause`.

This technique is widespread is commonly used.

Revision #6
Created 2022-12-28 17:39:36 UTC by Tamarine
Updated 2022-12-28 22:57:53 UTC by Tamarine