LESSON 11
ERROR HANDLING

Invincible Awareness

Wholeness: No matter how great we are at programming, sometimes our scripts have errors. They may occur because of our mistakes, an unexpected user input, an erroneous server response and for a thousand other reasons. Modern programming languages provide special mechanisms to handle runtime errors in a way that keeps programs as simple as possible and protects the experience of users when errors occur. **Science of Consciousnes:** In the relative world there will always be external influences and disruptions. It is a common experience that such influences are easier to handle if one is rested, calm, organized and alert.

Main Points

1. try..catch
2. throw and finally
3. extending Error

try..catch

- No matter how great we are at programming, sometimes our scripts have errors.
  - our mistakes,
  - unexpected user input,
  - erroneous server response
  - thousands of other reasons.
- Usually, a script “dies” (immediately stops) in case of an error, printing it to console.

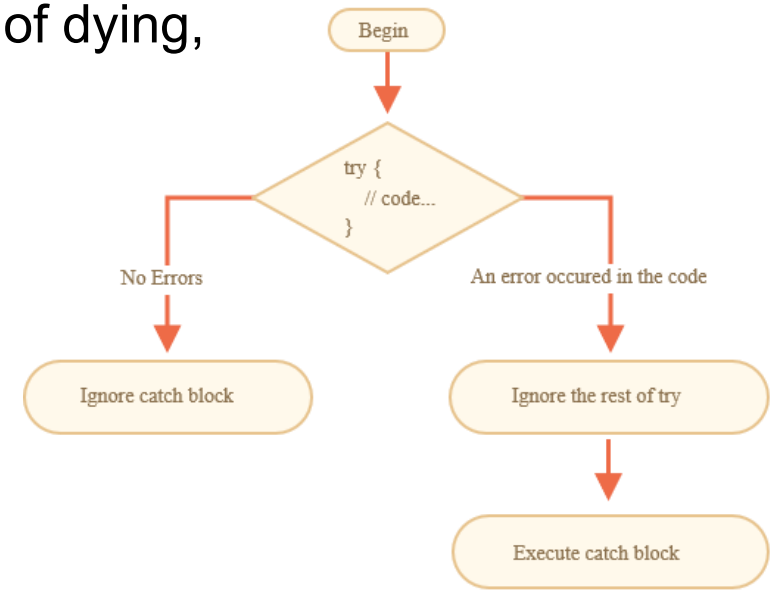
- syntax construct try..catch that allows to “catch” errors instead of dying,
  - do something more reasonable.

```
try {

  // code...

} catch (err) {

  // error handling
```



only works for runtime errors

- code must be runnable. In other words, it should be valid JavaScript.
- won't work if the code is syntactically wrong

```
try {
  {{{{{{{{{{{   // unmatched curly braces
} catch(e) {
  alert("The engine can't understand this code, it's invalid");
}
```

- JavaScript engine first reads the code, and then runs it.
  - errors that occur on the reading phrase are called "parse-time" errors
    - are unrecoverable
  - try..catch can only handle errors that occur in valid code.
    - called "runtime errors" or, sometimes, "exceptions"

Error object

- When an error occurs, JavaScript generates an object containing the details about it. The object is then passed as an argument to catch:

```
try {
  // ...
} catch(err) { // <-- the "error object", could use another word instead of err
  // ...
}
```

- For all built-in errors, the error object has two main properties:
  - name: Error name. For instance, for an undefined variable that's "ReferenceError".
  - Message: Textual message about error details
  - other non-standard properties available in most environments
    - Current call stack: string with sequence of nested calls that led to the error.

real-life use case of try..catch object

- JSON.parse(str) method to read JSON-encoded values.
- If json is malformed, JSON.parse generates an error, so the script “dies”.
- if something’s wrong with the data, the visitor will never know
  - unless open the developer console
  - users dislike when something “just dies” without message

```
let json = "{ bad json }";
try {
  let user = JSON.parse(json); // <-- when an error occurs...
  alert( user.name ); // doesn't work
} catch (e) {
  // ...the execution jumps here
  alert( "Our apologies, the data has errors, we'll try to request it one more time." );
  alert( e.name );
  alert( e.message );
}
```

- Here use the catch block only to show the message, but can do much more:
  - send a new network request,
  - suggest an alternative to the visitor,
  - send information about the error to a logging facility, … . All much better than just dying.

Examples

- See example *handling_exception1.js*, *handling_exception2.js*

works synchronously

- If an exception happens in “scheduled” code, like in setTimeout, then try..catch won’t catch it:
  - function itself is executed later,
  - when the engine has already left the try..catch construct.

```
try {
  setTimeout(function() {
    noSuchVariable; // script will die here
  }, 1000);
} catch (e) {
  alert( "won't work" );
}
```

- To catch an exception inside a scheduled function, try..catch must be inside that function:

```
setTimeout(function() {
  try {
    noSuchVariable; // try..catch handles the error!
  } catch {
    alert( "error is caught here!" );
  }
}, 1000);
```

See example: async_exception.js

Main Point Preview: try .. catch

Modern programming languages support exception handling with try..catch blocks to recover from or handle runtime errors. When a runtime error occurs in a try block, execution immediately jumps to the catch block to handle the error instead of the program halting with no explanation or recovery. Science of Consciousness: try .. Catch blocks allow programs to more effectively respond to and recover from error conditions. When we are aware, we can respond consciously to the worldly situations.

Main Point Preview: throw own errors

We can use the throw statement in our own code to jump control to a catch block in the event our program reaches some unexpected condition that the normal code is not designed to handle. In such cases we often want to define our own error classes with information about the specific problem encountered.

Throwing our own errors

- throw operator generates an error

  throw <error object>

- catch became a single place for all error handling: both for JSON.parse and other cases.

```
let json = '{ "age": 30 }'; // incomplete data
try {
  let user = JSON.parse(json); // <-- no JSON error, but suppose name is an assumed property
  if (!user.name) {
    throw new Error("Incomplete data: no name"); // (*)
  }
  alert( user.name );
} catch(e) {
  alert( "JSON Error: " + e.message ); // JSON Error: Incomplete data: no name
}
```

See example: throwing_err1.js

Error constructor

- JavaScript has many built-in constructors for standard errors:
  - Error, SyntaxError, ReferenceError, TypeError and others.
  - can use them to create error objects as well.
- For built-in errors name property is name of the constructor
  - message is taken from the argument.

```
let error = new Error("Things happen o_O");
alert(error.name); // Error
alert(error.message); // Things happen o_O
```

Catch should only process errors it knows

- Catch should only process errors it knows and “rethrow” all others.
  - In catch(err) {...} block analyze error object
  - If don’t know how to handle it then throw

```
let json = '{ "age": 30 }'; // incomplete data, missing name property
try {
  let user = JSON.parse(json);
  if (!user.name) {
    throw new SyntaxError("Incomplete data: no name");
  }
  blabla(); // unexpected error
  alert( user.name );
} catch(e) {
  if (e.name == "SyntaxError") {
    alert( "JSON Error: " + e.message );
  } else {
    throw e; // rethrow (*)  //must be caught by outer try..catch or kills script (runtime exception)
  }}
```

See example: throwing_err2.js

Catch in surrounding try..catch

- Here readData only knows how to handle SyntaxError,
  - outer try..catch handles everything else

```
function readData() {
  let json = '{ "age": 30 }';
  try {
    // ...
    blabla(); // error!
  } catch (e) {
    // ...
    if (e.name != 'SyntaxError') {
      throw e; // rethrow (don't know how to deal with it)
    }
  }
}

try {
  readData();
} catch (e) {
  alert( "External catch got: " + e ); // caught it!
```

try..catch..finally

- If it exists, it runs in all cases:
  - after try, if there were no errors,
  - after catch, if there were errors.

See example: finally.js

finally and return

- finally clause works for any exit from try..catch. That includes an explicit return.

- In the example below, there's a return in try.
  - finally is executed just before the control returns to the outer code.

```
function func() {
  try {
    return 1;
  } catch (e) {
    /* ... */
  } finally {
    alert( 'finally' );
  }
}
alert( func() ); // first works alert from finally, and then this one
```

See example: finllay2.js

Custom errors, extending Error

- often need our own error classes to reflect specific things that may go wrong in our tasks
- Good idea to inherit the functionality of existing error class, such as Error

See example: custom_error.js

Exercises

- Exercises from https://eloquentjavascript.net/08_error.html

Main Point Preview: throw own errors

We can use the throw statement in our own code to jump control to a catch block in the event our program reaches some unexpected condition that the normal code is not designed to handle. In such cases we often want to define our own error classes with information about the specific problem encountered.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

Invincible awareness

1. Programs in the world must deal with unexpected error conditions in a graceful way.

2. The try..throw..catch..finally exception handling mechanism is used in most modern languages as a way to effectively handle runtime errors and also keep program logic uncluttered with error handling code.

---

3. **Transcendental consciousness**. Is the experience of perfectly still pure wakefulness that is unaffected by external stimuli. “Noise is no barrier to meditation.”

4. **Impulses within the transcendental field:** Thoughts from deep levels of awareness quickly and spontaneously find alternatives to any blocks.

5. **Wholeness moving within itself:** In unity consciousness, one experiences life in terms of the invincible force of evolution and progress to fulfillment.