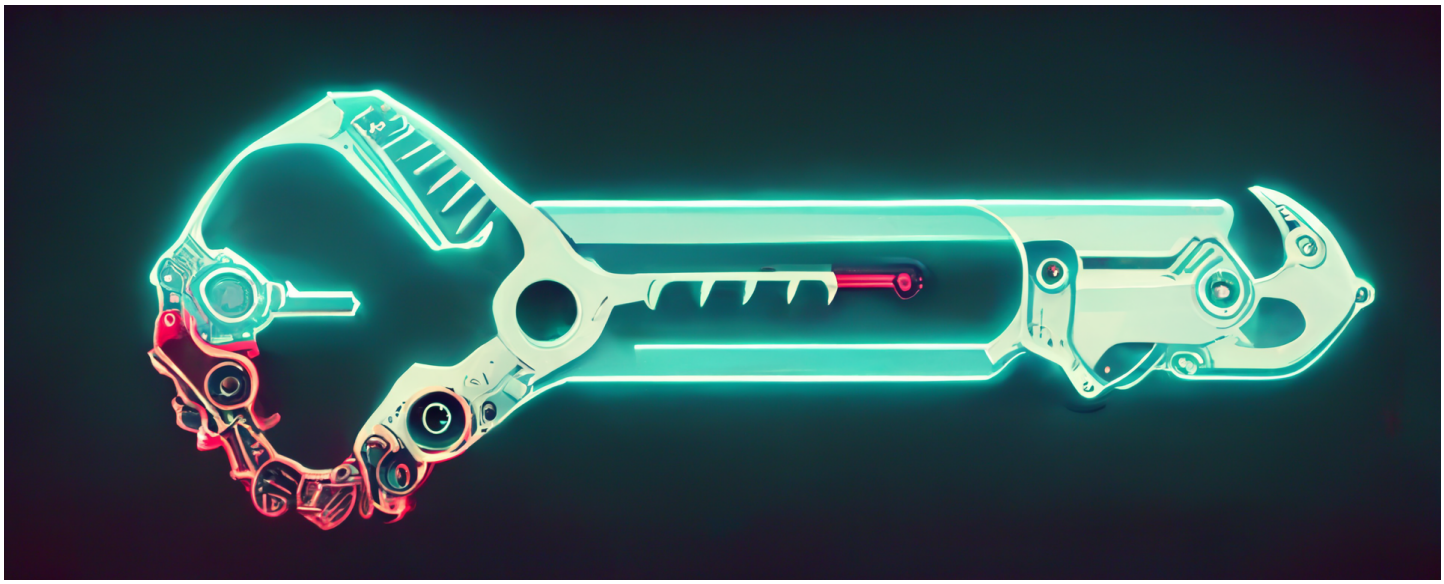
9.0 JS try catch Statement



cyberpunk key - answered by Kristanna Agnes Nical

- When executing JavaScript code, **different errors can occur**.
- Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.
- In this example we misspelled `"alert"` as `"adddlert"` to deliberately produce an error:

```
<p id="demo"></p>

<script>
try {
  adddlert("Welcome guest!"); //This line will cause an error when the code is ran
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

Trying and catching errors

- The `try` statement allows you to define a block of code to be tested for errors while it is being executed.
- The `catch` statement allows you to define a block of code to be executed, if an error occurs in the `try` block.

Keyword	Description	Sample code	Code explanation

`try`	• Used to enclose a block of code that may potentially throw an exception.	`function juiceFruit(fruit) {`
		`  try {`
		`  if (fruit === "apple") {`
		`  throw new Error("Oops! Apples can't be juiced.");`
		`  }`
		`  console.log("Juicing the " + fruit + "...");`
		`  } catch (error) {`
		`  console.error("An error occurred: " + error.message);`
		`  } finally {`
		`  console.log("Cleaning the juicer and countertop.");`
		`  }`
		`}`
		
		`juiceFruit("banana");`
		`juiceFruit("apple");`
		`juiceFruit("orange");`
		
		`console.log("Finished juicing the fruits.");`
			• The `juiceFruit` function attempts to juice a fruit specified as an argument.
			• If the fruit is an "apple", it throws an exception with an error message.
			• The code logs messages for the juicing process and cleaning, catches and handles any exceptions, and finally, it logs that all fruits have been juiced when the function calls are made.

`catch`	• Used within a `try`...`catch` block to define a section of code that handles and processes exceptions thrown in the associated `try` block.		
`throw`	• Used to manually raise or "throw" an exception.		
`finally`	• Used within a `try`...`catch` block to specify a section of code that is executed regardless of whether an exception is thrown in the `try` block or not.		

Throwing errors manually

- When an error occurs, JavaScript will normally stop and generate an error message.
  - This is called "throwing an exception" (throw an error).
- The `throw` statement allows you to create a custom error.
  - The exception can be a JavaScript String, a Number, a Boolean or an Object:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

- If you use throw together with `try` and `catch`, you can control program flow and generate custom error messages.
- This example examines input. If the value is wrong, an exception (err) is thrown.
  - The exception (err) is caught by the `catch` statement and a custom error message is displayed:

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

- Modern browsers will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

```
<input id="demo" type="number" min="5" max="10" step="1">
```

Ignoring the result

- The `finally` statement lets you execute code, after `try` and `catch`, regardless of the result:

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

```
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Error: " + err + ".";
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```

The Error Object

- JavaScript has a built in error object that provides error information when an error occurs.
- The error object provides two useful properties: `name` and `message`.

Property	Description	Sample code	Code explanation
`.name`	• Sets or returns an	`function divideNumbers(a, b) {` `  try {`	• The `divideNumbers` function attempts to

	error name	`if (b === 0) {`
		`throw new Error("Division by zero is not allowed.");`
		`}`
		`return a / b;`
		`} catch (error) {`
		`console.error("Error Name: " + error.name);`
		`console.error("Error Message: " + error.message);`
		`}`
		`}`
		
		`const result1 = divideNumbers(10, 2);`
		`const result2 = divideNumbers(8, 0);`
			divide two numbers (`a` and `b`).
			• If `b` is equal to 0, it throws an `Error` with the message "Division by zero is not allowed."
			• When the function is called with different arguments, it calculates the division in the first case (`result1`) and catches and logs the error's name and message in the second case (`result2`) due to the division by zero.

`.message`	• Sets or returns an error message (a string)		

Error Name Values

- Six different values can be returned by the error `name` property:

Error Name	Description	Sample code	Why did the error occur?

`RangeError`	• Thrown if you use a number that is outside the range of legal values.	`let num = 1;`
		`try {`
		`num.toPrecision(500);`
		`}`
		`catch(err) {`
		`alert(err.name);`
		`}`
			• You cannot set the number of significant digits of a number to 500.

`ReferenceError`	• Thrown if you use (reference) a variable that has not been declared.	`let x = 5;`
		`try {`
		`x = y + 1;`
		`}`
		`catch(err) {`
		`alert(err.name);`
		`}`
			• `y` is not declared.

`SyntaxError`	• Thrown if you try to evaluate code with a	`try {` `alert('Hello);`	• Missing `'`.

	syntax error.	`}`
		`catch(err) {`
		`  alert(err.name);`
		`}`	

`TypeError`	• Thrown if you use a value that is outside the range of expected types.	`let num = 1;`
		`try {`
		`  num.toUpperCase();`
		`}`
		`catch(err) {`
		`  alert(err.name);`
		`}`
			• You cannot convert a number to upper case.

Additional Material

- **Learn more**
  - W3Schools