Promises & Error Handling

Topics Covered:

- Promises
- Error Handling

Topics in Detail:

Promises

- **Asynchronous operations** in JavaScript are **handled** using **promises**.
- **Multiple asynchronous operations** can make the code unmanageable by creating **callback hell**. Promises can **easily manage** this situation.
- **Events** and **callback** functions are used to **handle asynchronous operations** before promises, but they had **limited functionalities** making the **code unmanageable**.
- **Callback hell** created by **multiple callback functions** make the **code unmanageable**.
- **Multiple callbacks** at the **same time** are **not easy** to **handle**.
- **Promises** can **handle multiple asynchronous operations** easily.
- They can handle multiple callbacks at the same time, **avoiding a callback hell** situation.
- Promises **improve** the code **readability** in the most effective and efficient manner.

Benefits of promises

- Improves **Code Readability**
- Better handling of **asynchronous operations**
- Better **flow of control** definition in **asynchronous logic**
- Better **Error Handling**

States of Promises

1. **fulfilled**: Promise is **succeeded**
2. **rejected**: Promise is **failed**
3. **pending**: Promise is **still pending,** i.e. not fulfilled or rejected yet
4. **settled**: Promise has **fulfilled** or **rejected**

Create a promise using the promise constructor

Syntax

```
var promise = new Promise(function(resolve, reject){
     //do something
});
```

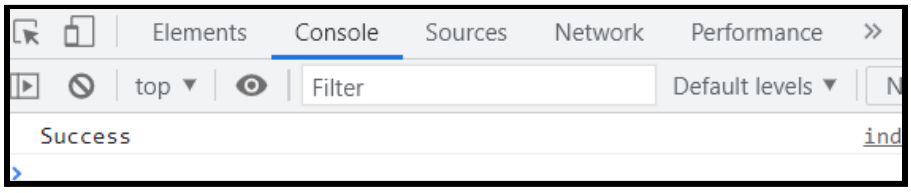
Parameters

- **The promise constructor** can have only **one argument,** which is a **callback function**.
- The callback function can take **two arguments**
    - resolve
    - reject
- If the operations inside a **callback function performed** well, then call **resolve**.
- If **not performed** well then call **reject**.

```
<!DOCTYPE html>
<html>
<body>
<script>
var promise = new Promise(function(resolve, reject) {
const x = "JS Promises";
const y = "JS Promises";
if(x === y) {
  resolve();
} else {
  reject();
}
});

promise.
  then(function () {
    console.log('Success');
  }).
  catch(function () {
    console.log('Some error has occurred');
  });
</script>
</body>
</html>
```

Output



Promise Consumers

- Promises can be consumed by using **.then** and **.catch** methods.
- **then()**
- When a promise is either **resolved or rejected,** then() is invoked. It acts as a career taking **data from the promise** and further **executes** it successfully.

Parameters

- **then()** has two functions as parameters
- If the promise is **resolved** and the **result is received**, then the **first function** will be executed.
- If the promise is **rejected** and an **error is received**, then the **second function** will be executed.
- **Syntax**

```
.then(function(result){
        //handle success
    }, function(error){
        //handle error
    })
```

- **catch():** When a promise is **either rejected** or if some **error** has occurred, **catch()** is invoked. If there is any chance of getting an error, it is used as an **Error handler**.

Parameters

- **then()** has one function as parameters
- If the promise is **rejected or the error** has occurred, then the function can handle it.

  Syntax

```
.catch(function(error){
        //handle error
    })
```

Applications

- To handle **asynchronous events**, promises are used.
- To handle **asynchronous http requests**, promises are used.

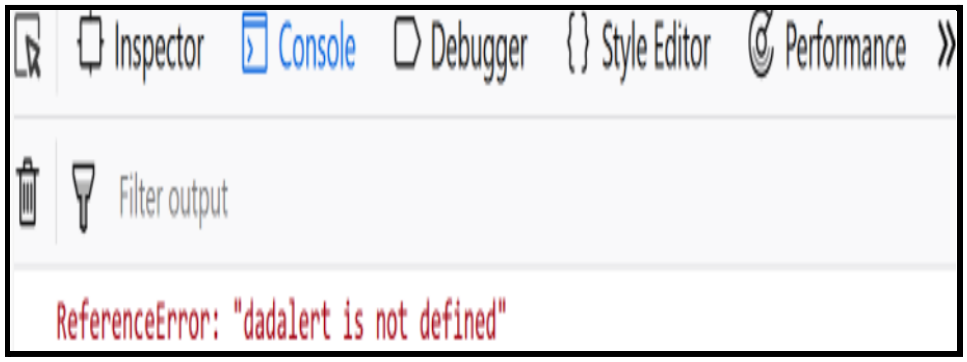
Error Handling

- **Errors** will definitely occur while **executing JavaScript code**.
- Error can occur in the following situations
  - When there is a **fault from the programmer side**
  - When the **input is wrong**
  - When there is a **problem with the logic** of the program
- Using the below five statements, we can solve the errors
  - **try** - **Check for errors** in a block of code.
  - **catch** - **Handles the error** if there are any
  - **throw** - lets you make your **own error**
  - **finally** - Execute the code **after try and catch**
- This block of code will run **regardless of the result** of the **try-catch block**.
- **Example**

```
try {
  dadalert("Welcome Fellow Geek!");
}
catch(err) {
  console.log(err);
}
```

- **Dadalert** is not a reserved keyword and neither it is defined, hence we get an error.

Output



Try and catch block

- The try statement will let you **check** whether there is an **error** in a specific block of code.
- The catch statement will **display the error** if there is any in the **try block**.
- **Syntax**

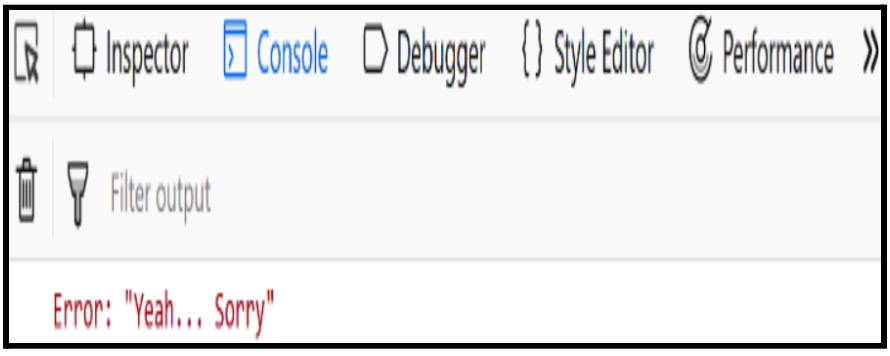
```
try {
 Try Block to check for errors.
}
catch(err) {
  Catch Block to display errors.
}
```

Throw

- JavaScript will stop and **generate an error message** when any **error** occurs. The **throw** statement will allow us to create any **custom-made errors**.

```
try {
    throw new Error('Yeah... Sorry');
}
catch(e) {
    console.log(e);
}
```

- **Output**



Finally Block

- After the execution of **try/catch block**, the **finally block** runs **unconditionally.**
- **Syntax**

```
 try {
 Try Block to check for errors.
}
catch(err) {
  Catch Block to display errors.
}
finally {
  Finally Block executes regardless of the try / catch result.
}
```

- **Example**

```
try {
  alert( 'try' );
} catch (e) {
  alert( 'catch' );
} finally {
  alert( 'finally' );
}
```

- **Output**