CYDEO

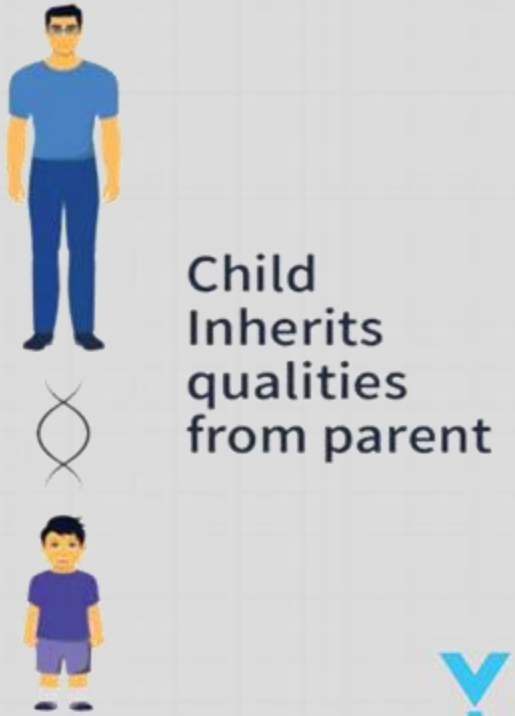
JavaScript Programming Day03

Content

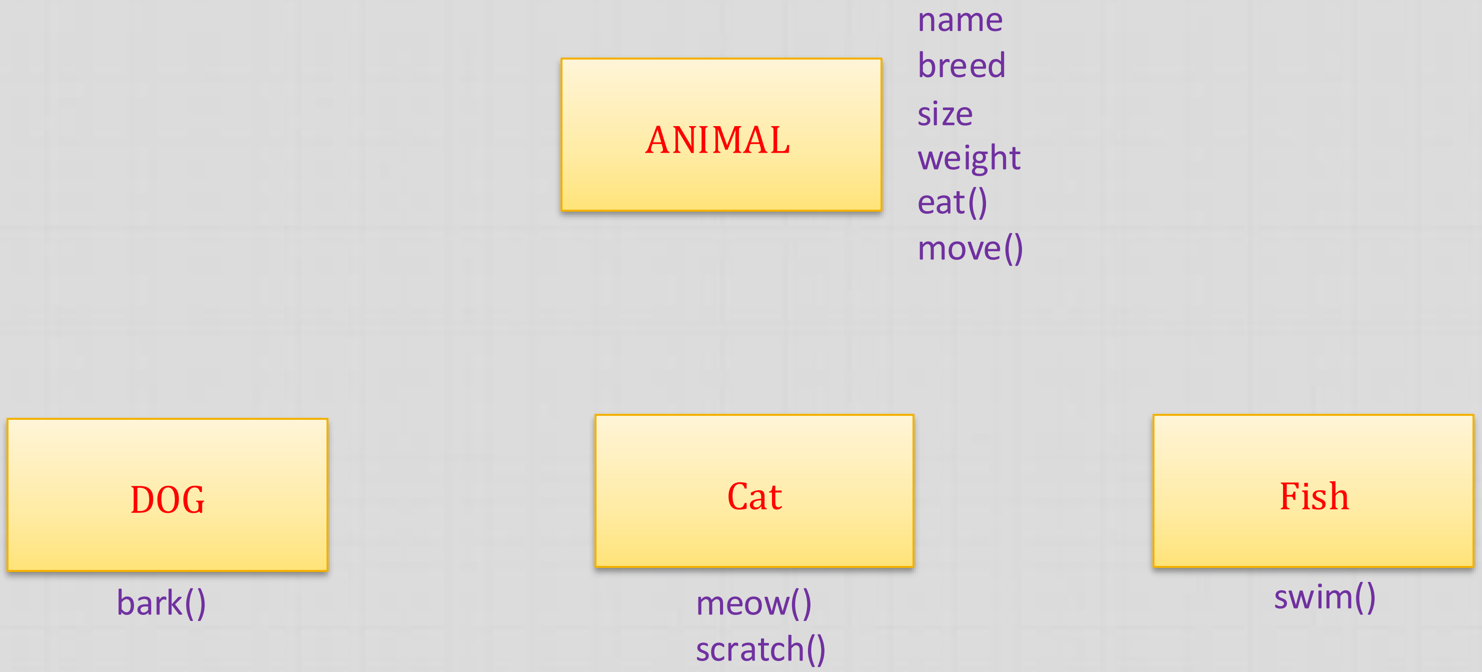
- Class & Objects
- Inheritance
- Exceptions/Errors
- Promises
- Async & Await

Inheritance

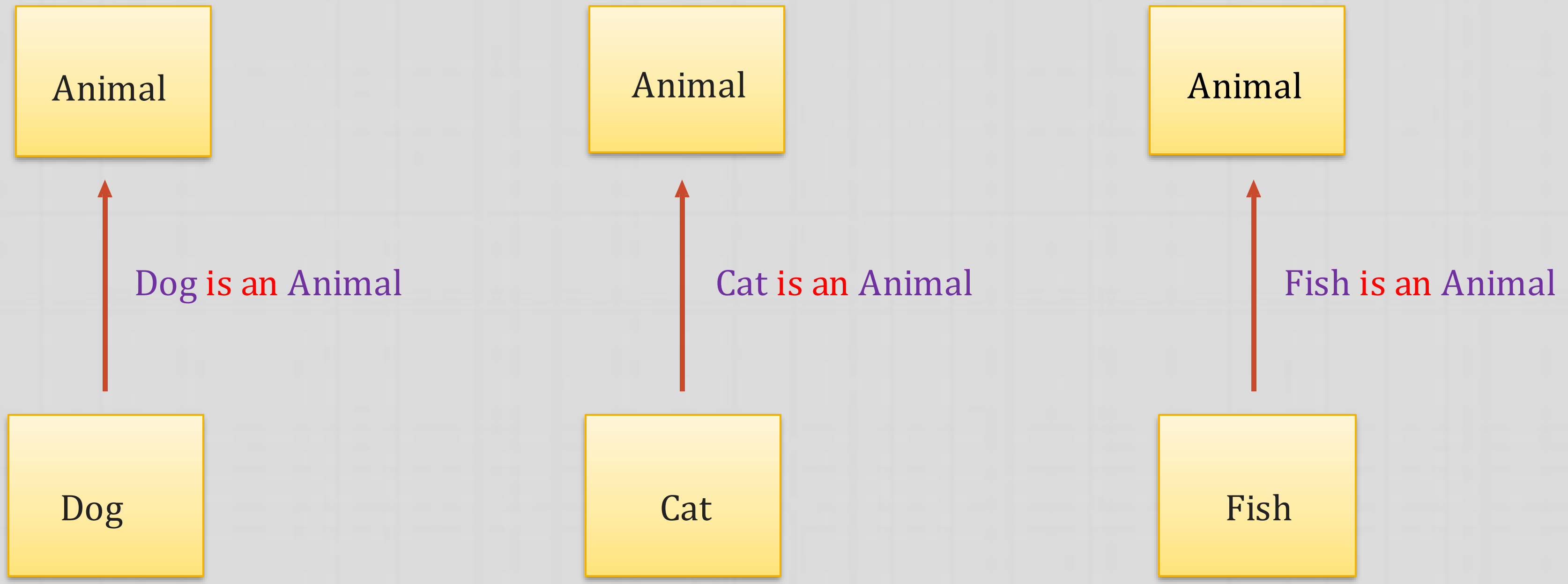
- Used for creating Is A relationship among the classes
- Allows one class to inherit the variables and methods from another class



Inheritance



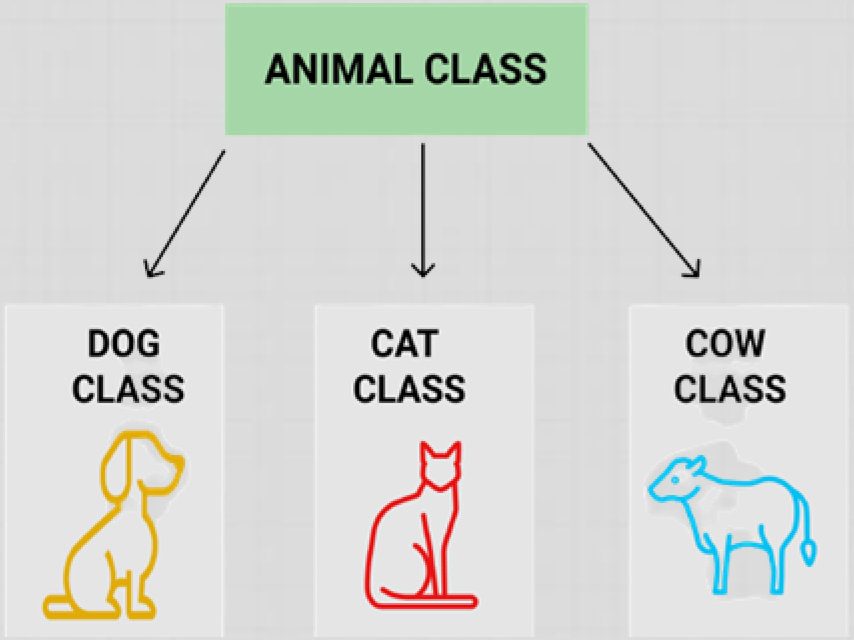
Inheritance

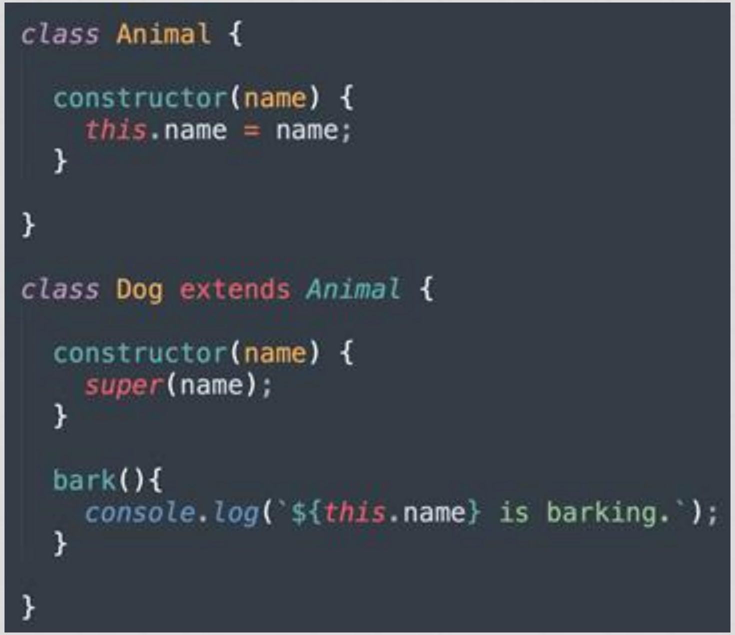


The animal is called SUPER class and the other classes are called SUB class

Inheritance

- A class can inherit from one parent class by specifying the parent class name after the extends keyword.
- The constructor and private members can not be inherited from parent to child

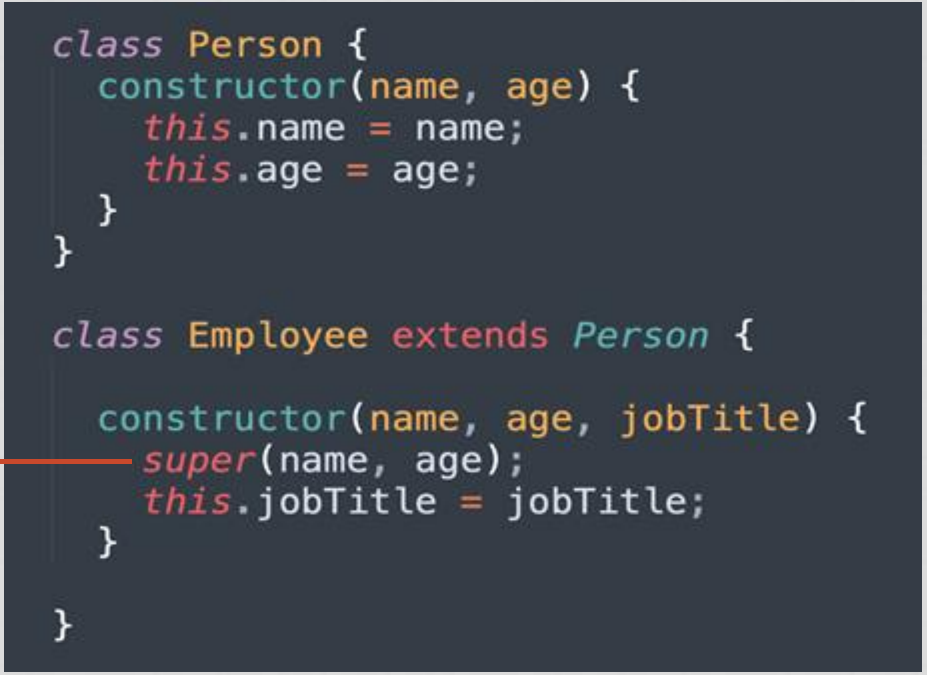




```
class Animal {

  constructor(name) {
    this.name = name;
  }

}

class Dog extends Animal {

  constructor(name) {
    super(name);
  }

  bark(){
    console.log(`${this.name} is barking.`);
  }

}
```

Super keyword

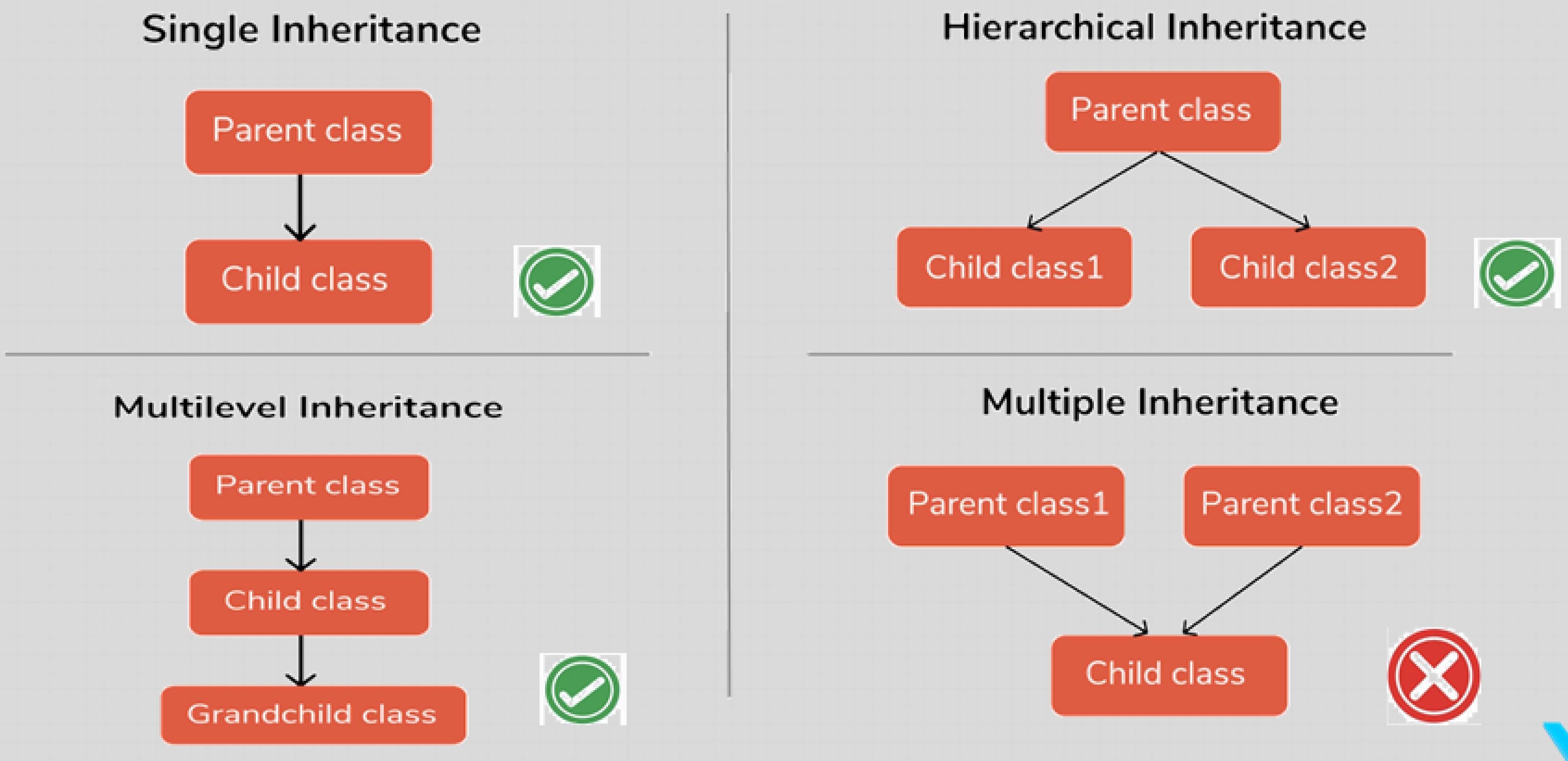
- The super keyword refers to the superclass (Parent).
- We can use super() to call a superclass's constructor



```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}

class Employee extends Person {

  constructor(name, age, jobTitle) {
    super(name, age);
    this.jobTitle = jobTitle;
  }

}
```

Accesses the parent class members

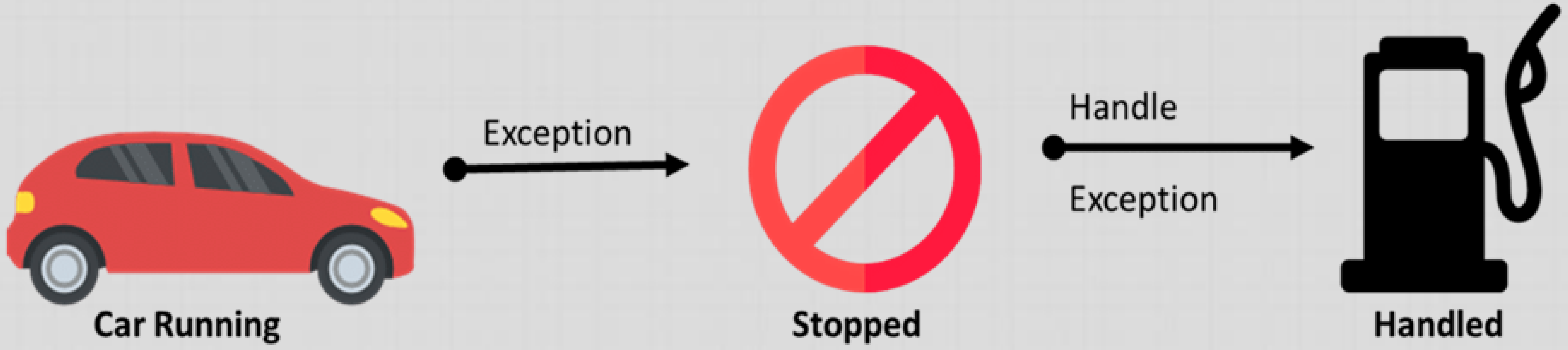
Types of Inheritance



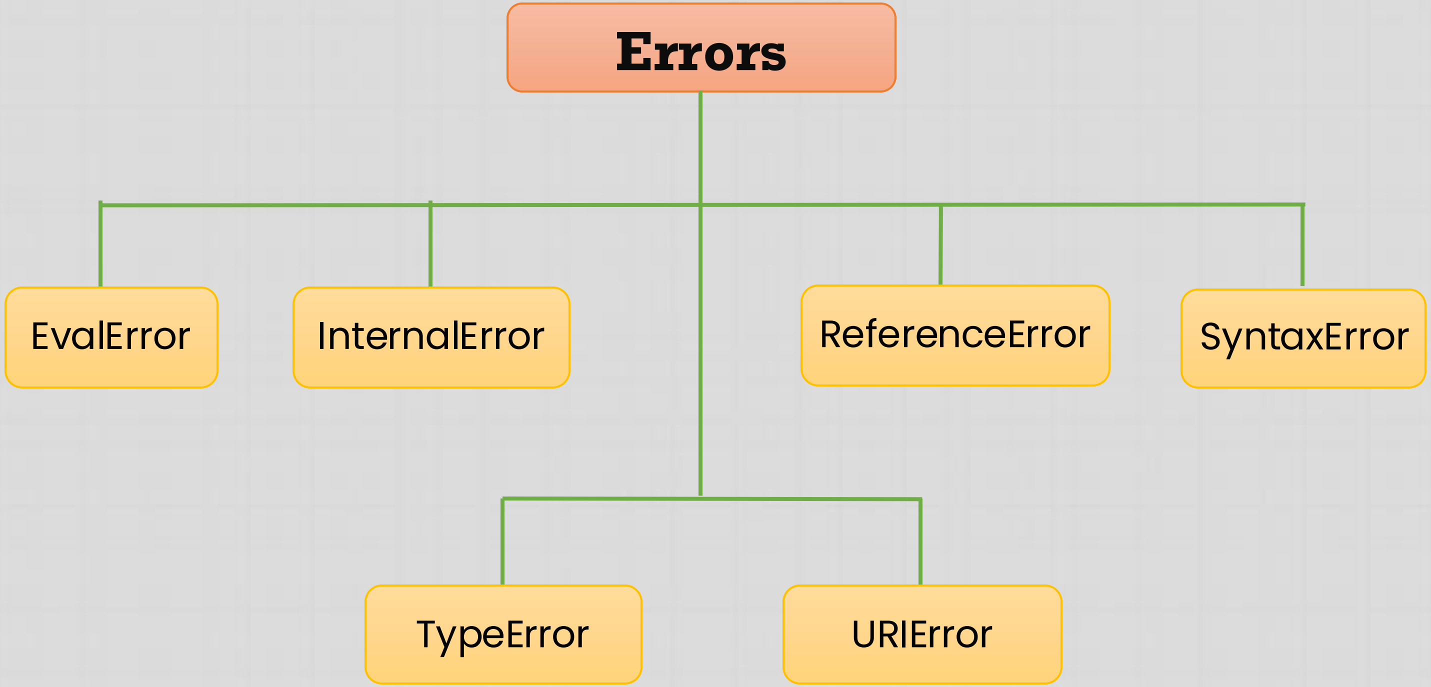
Exceptions/Errors

Exceptions/Errors

- An unwanted or unexpected event (Something went wrong)
- Exceptions/error are runtime errors that disrupt the normal flow of code execution

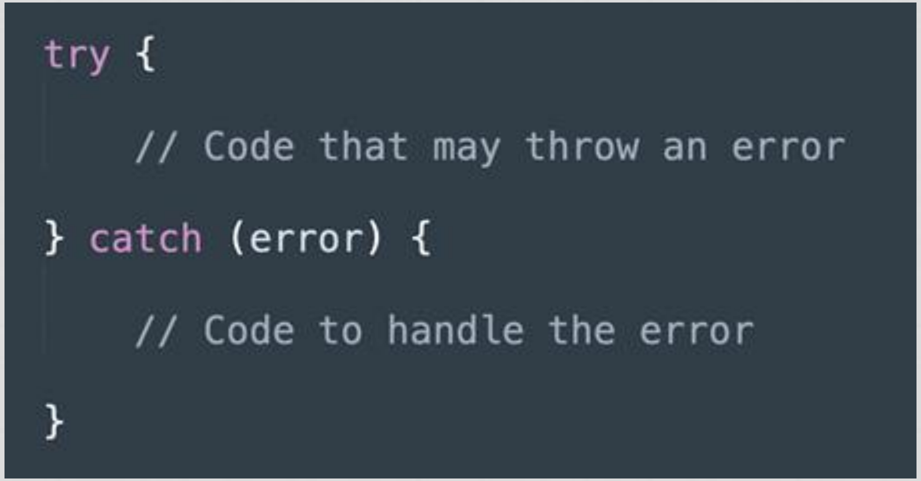


Exceptions/Errors Hierarchy



Try & Catch Blocks

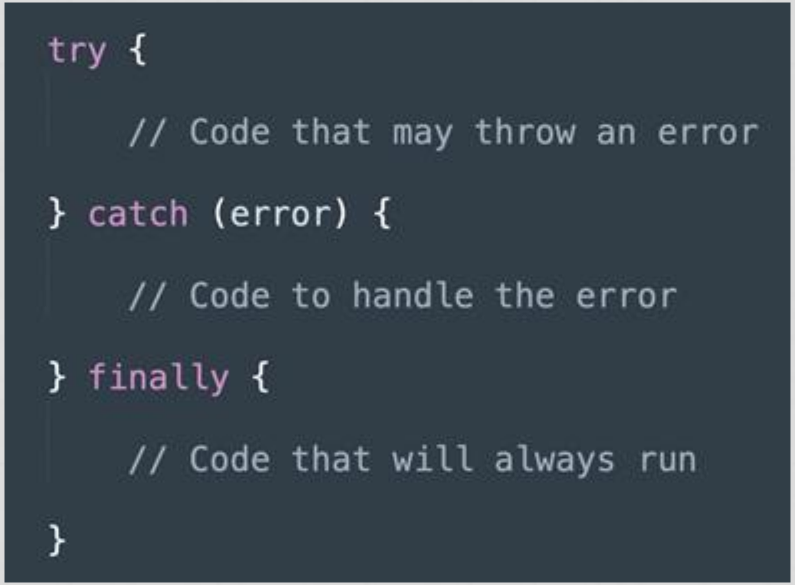
- Exceptions are handled using try-catch blocks.



```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
}
```

Finally Block

- An optional block that can be given after the last catch block
- Always executed after try & catch blocks whether an exception/error occurs or not



```
try {
    // Code that may throw an error
} catch (error) {
    // Code to handle the error
} finally {
    // Code that will always run
}
```

Throw Keyword

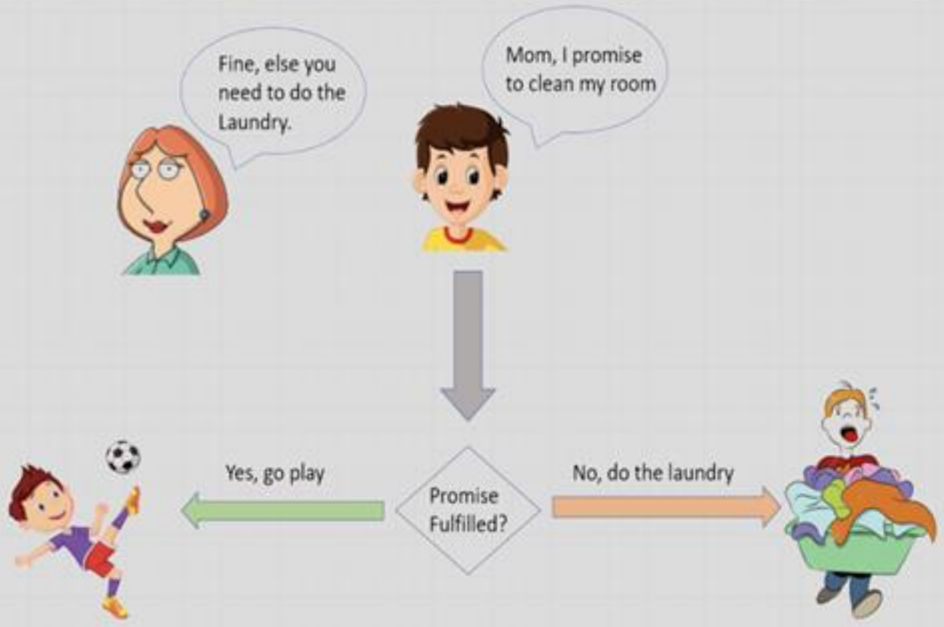
- Used for manually throwing an exception

```
throw new Error("Error Message");
```

Promises

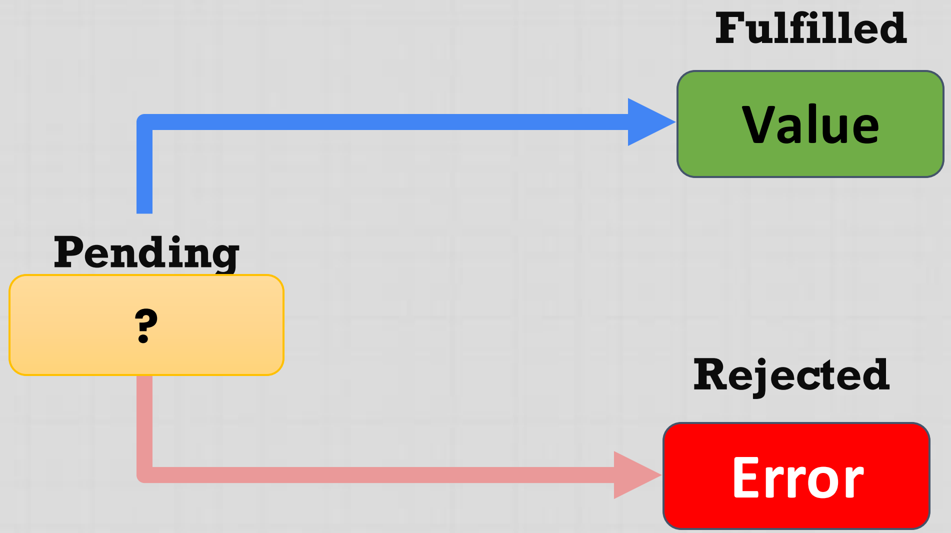
Promises

- Promises in JavaScript can Help handle async operations
- Promises are objects that represent the eventual completion (or failure) of an async operation and its resulting value
- Promises Wait for operations to finish

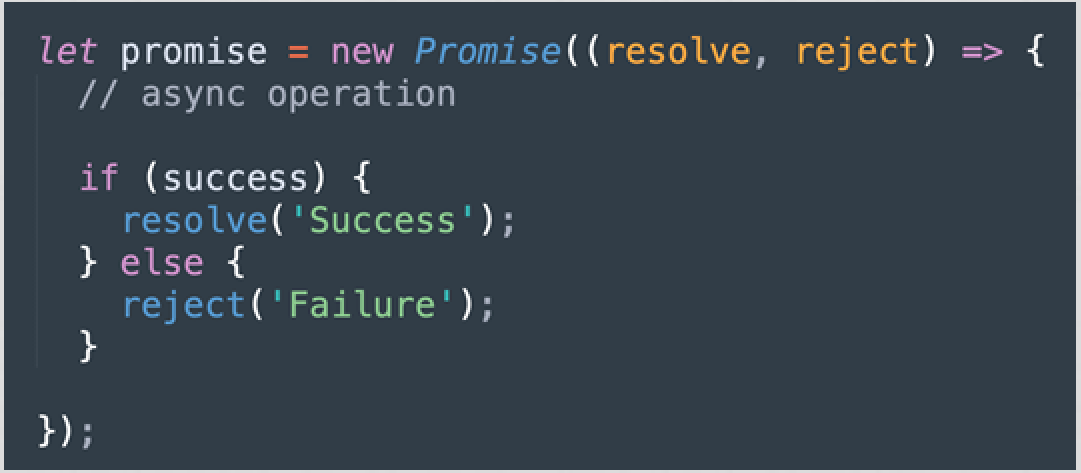

Fine, else you need to do the Laundry.
Mom, I promise to clean my room
Yes, go play
Promise Fulfilled?
No, do the laundry

Promise States

- A Promise in JavaScript has three states:
  - Pending: Initial state, waiting
  - Fulfilled: Operation completed successfully
  - Rejected: Operation failed



Promise Creation

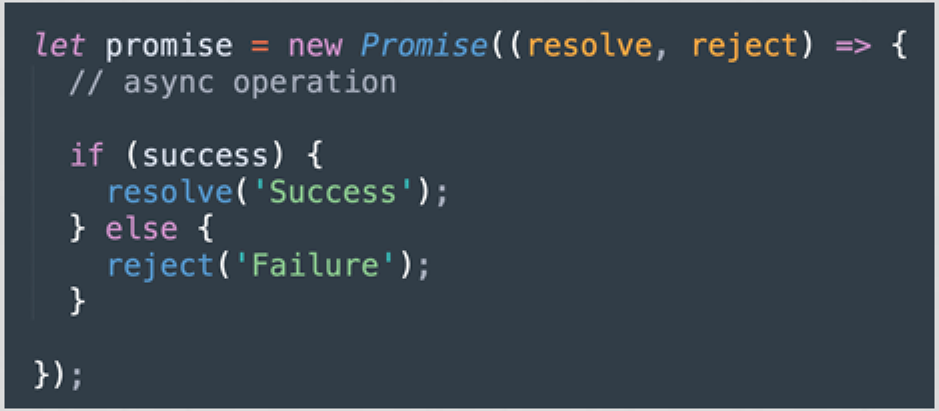


```
let promise = new Promise((resolve, reject) => {
  // async operation

  if (success) {
    resolve('Success');
  } else {
    reject('Failure');
  }

});
```

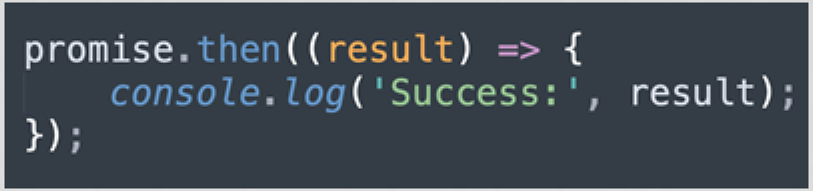
- Resolve: Call if operation succeeds
- Reject: Call if operation fails

Handling Promises

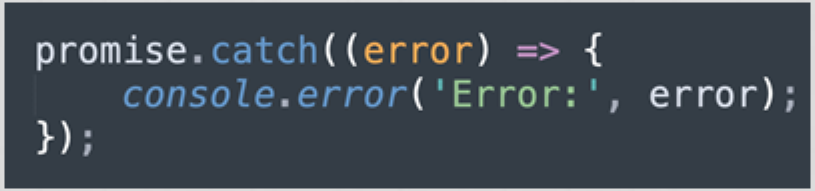
- A Promises is handled by using then() and catch() methods of the promise objects
  - then(): Runs if promise is fulfilled
  - catch(): Runs if promise is rejected
  - finally(): Optional to call and always runs whether the promise is fulfilled or rejected



```
let promise = new Promise((resolve, reject) => {
  // async operation

  if (success) {
    resolve('Success');
  } else {
    reject('Failure');
  }

});
```



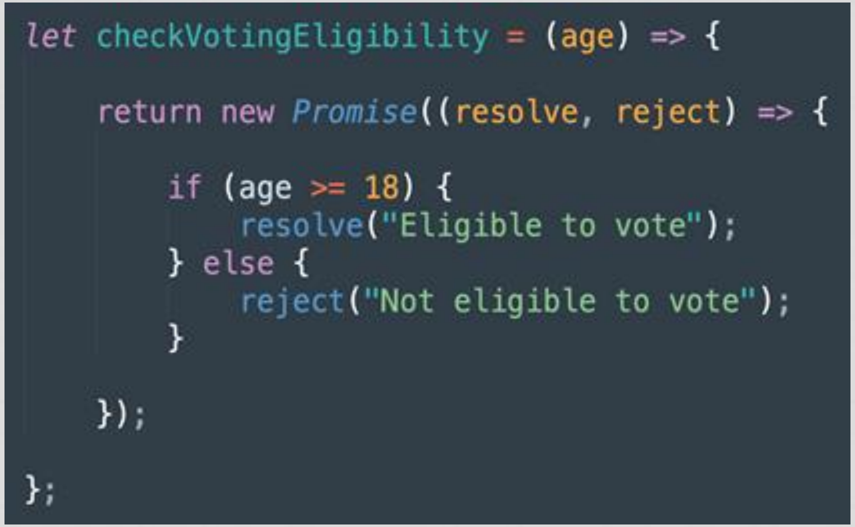
```
promise.then((result) => {
    console.log('Success:', result);
});
```



```
promise.catch((error) => {
    console.error('Error:', error);
});
```

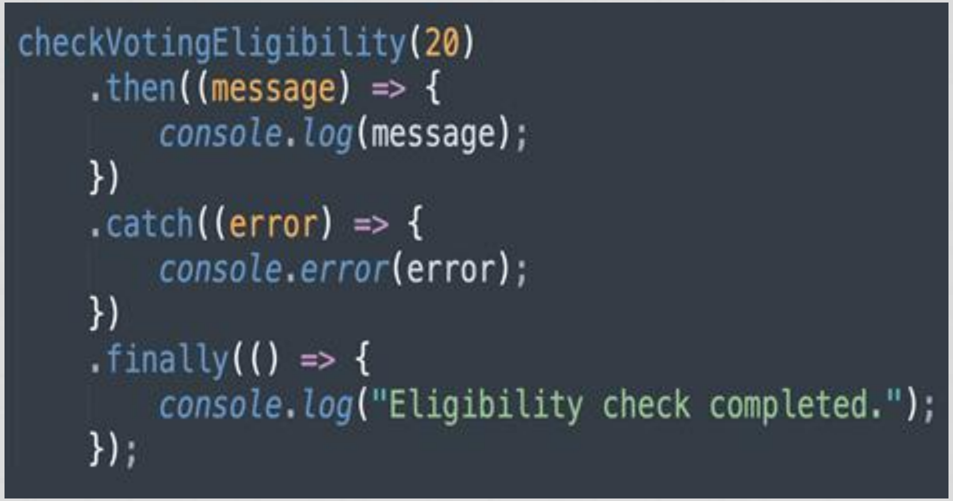
Promise Example: Voting Eligibility Check

Creating a promise object



```
let checkVotingEligibility = (age) => {

    return new Promise((resolve, reject) => {

        if (age >= 18) {
            resolve("Eligible to vote");
        } else {
            reject("Not eligible to vote");
        }

    });

};
```

Handling the promise

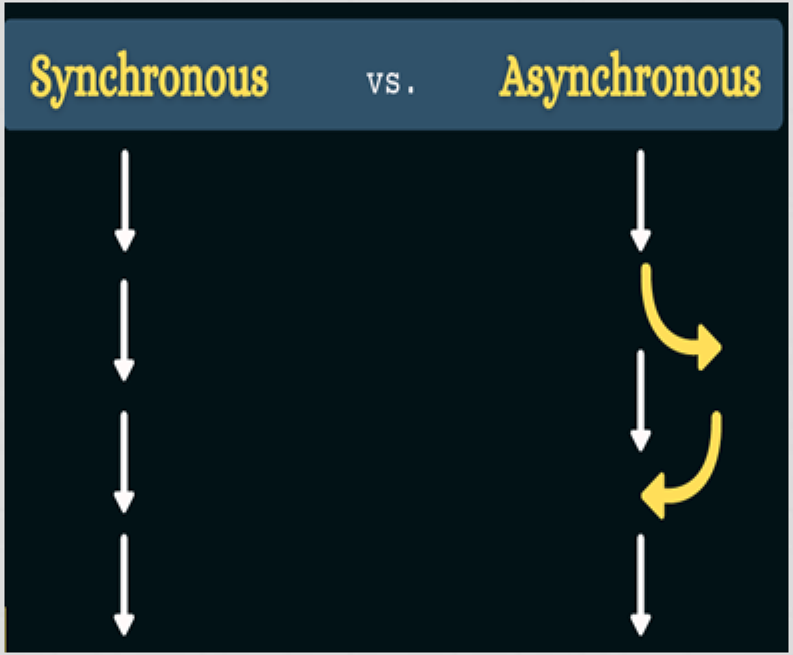


```
checkVotingEligibility(20)
    .then((message) => {
        console.log(message);
    })
    .catch((error) => {
        console.error(error);
    })
    .finally(() => {
        console.log("Eligibility check completed.");
    });
```

Async & Await

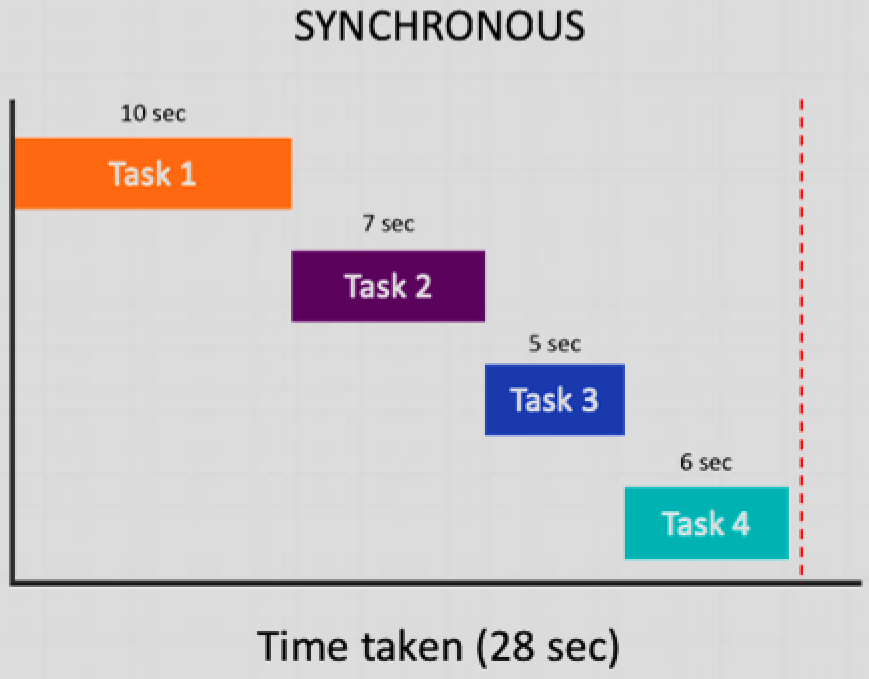
Asynchronous Functions

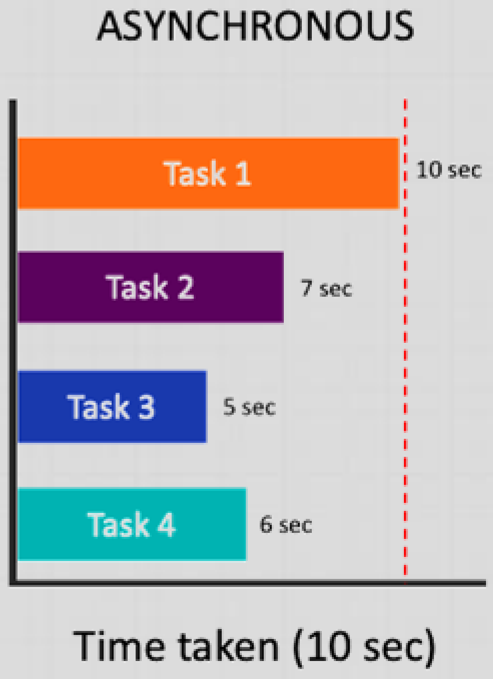
- Functions that operate asynchronously
- Allows the program to run other code while waiting for the time-consuming operations to complete
- Asynchronous functions don't block the execution of the rest of the code
- Asynchronous functions can be achieved using the async and await keywords



Benefits of Asynchronous Functions

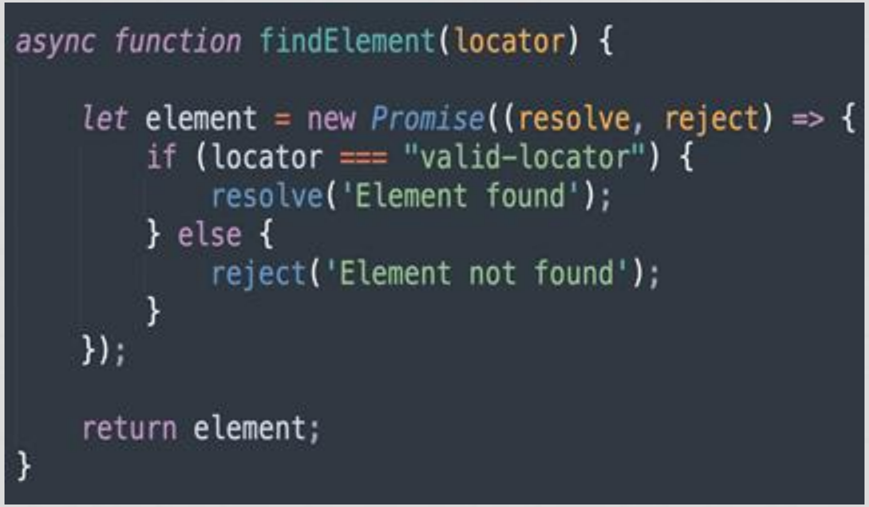
- Improves performance by not blocking the execution of other code
- Provides a better user experience by Improving application responsiveness



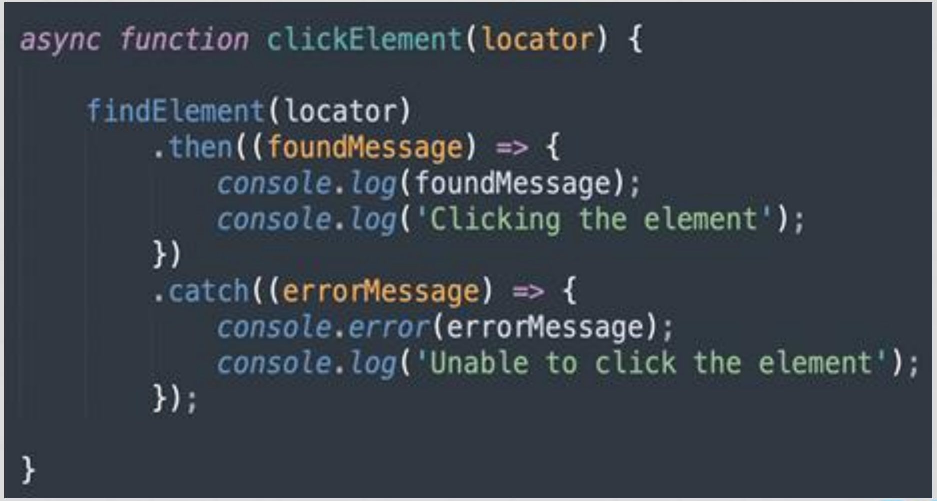


Async keyword

- Used to declare functions as asynchronous
- The function returns a promise, even if it doesn't explicitly return one



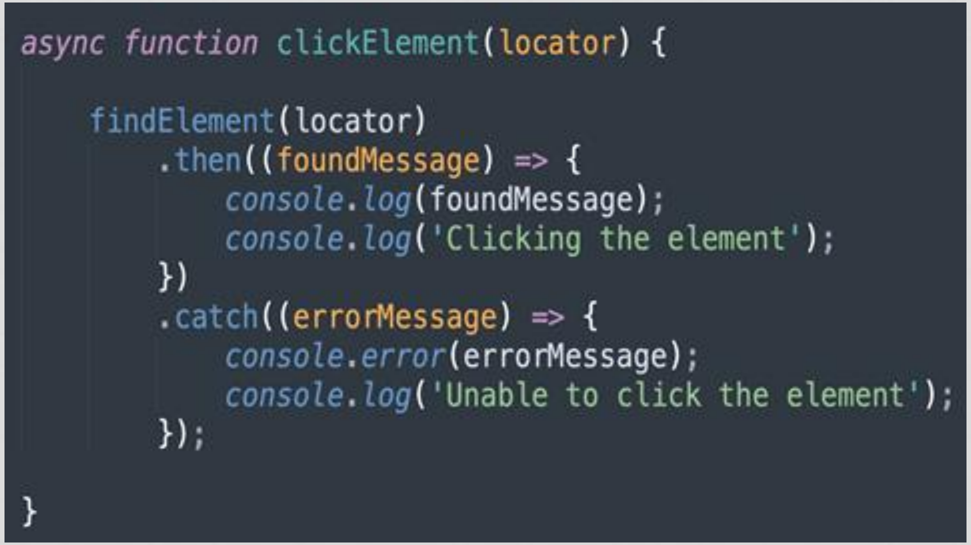
```javascript
async function findElement(locator) {

    let element = new Promise((resolve, reject) => {
        if (locator === "valid-locator") {
            resolve('Element found');
        } else {
            reject('Element not found');
        }
    });

    return element;
}
```



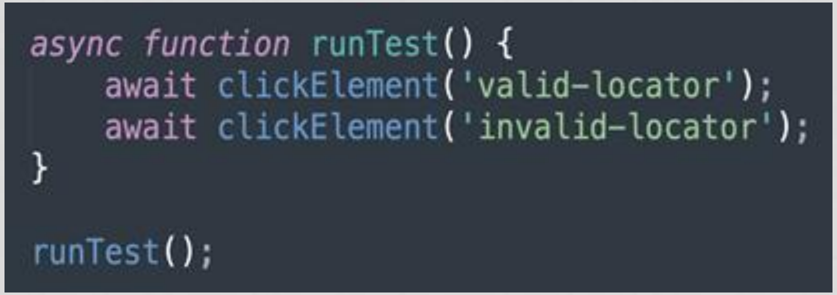
```javascript
async function clickElement(locator) {

    findElement(locator)
        .then((foundMessage) => {
            console.log(foundMessage);
            console.log('Clicking the element');
        })
        .catch((errorMessage) => {
            console.error(errorMessage);
            console.log('Unable to click the element');
        });

}
```

Await keyword

- Used inside an async function to wait for a promise
- Pauses the function until the promise is resolved or rejected

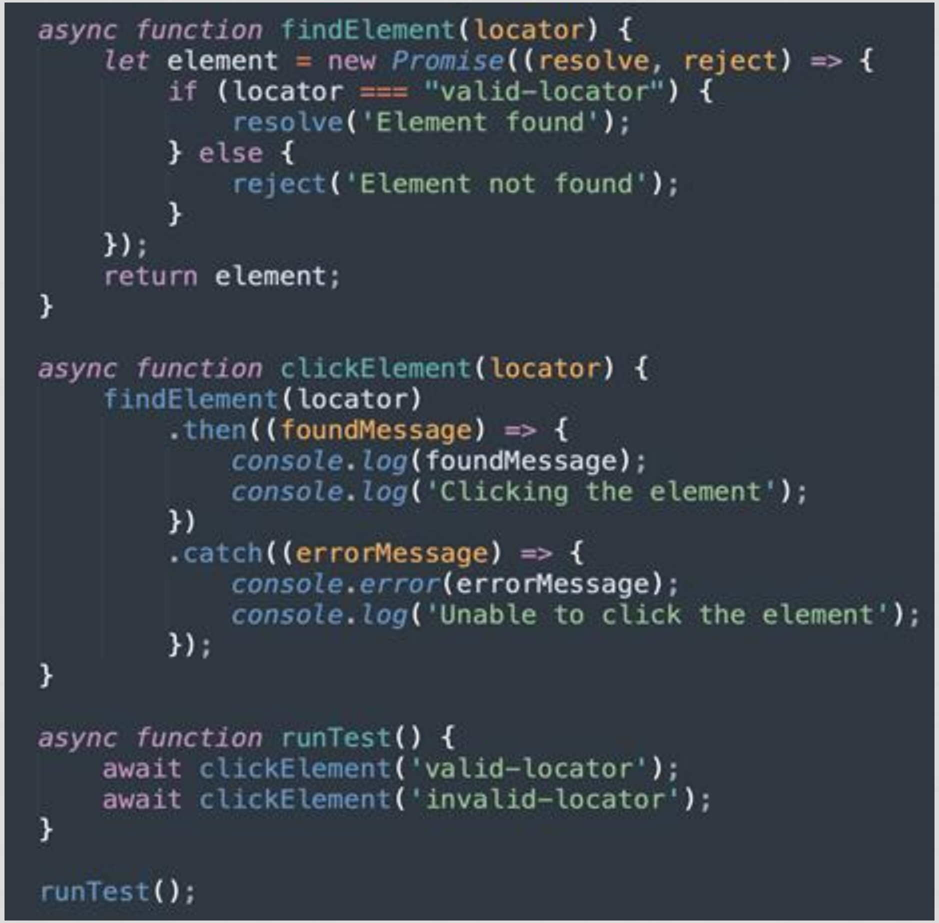


```
async function clickElement(locator) {

    findElement(locator)
        .then((foundMessage) => {
            console.log(foundMessage);
            console.log('Clicking the element');
        })
        .catch((errorMessage) => {
            console.error(errorMessage);
            console.log('Unable to click the element');
        });

}
```



```
async function runTest() {
    await clickElement('valid-locator');
    await clickElement('invalid-locator');
}

runTest();
```

Example Explanation



```
async function findElement(locator) {
    let element = new Promise((resolve, reject) => {
        if (locator === "valid-locator") {
            resolve('Element found');
        } else {
            reject('Element not found');
        }
    });
    return element;
}

async function clickElement(locator) {
    findElement(locator)
        .then((foundMessage) => {
            console.log(foundMessage);
            console.log('Clicking the element');
        })
        .catch((errorMessage) => {
            console.error(errorMessage);
            console.log('Unable to click the element');
        });
}

async function runTest() {
    await clickElement('valid-locator');
    await clickElement('invalid-locator');
}

runTest();
```

- findElement function:
  - Uses async to return a promise
  - Resolves if the locator is valid
  - Rejects if the locator is invalid
- clickElement function:
  - Uses await to wait for findElement
  - Logs message based on promise result
- runTest function:
  - Uses await to wait for both clickElement