Javascript Promises and Asynchronous Functions

A javascript promise is a piece of code that allows for communicating between client and server in a fluid manner. When using code that communicates with a server there is always time delay. A normal program without the use of promises or equivalent methods could possible run to completion before receiving the response from the server which would exhibit unexpected behavior. Promises allow for the code to pause as necessary in order to receive the response (usually some form of json or other object format). The code can then use the object response to continue with the code in a cohesive manner. As the client doesn't know the format of the servers response, promises have a two fold answer method, a fulfilled response or an error message. These are caught with the Promise.then() and Promise.catch() methods. While the code is waiting for a server response, any code not contained in the .then() clause can continue to run without interruption. When the servers response is obtained, whereon the code returns to complete the code in the .then().

Example:

```
let correct = true;
let cheeseType = new Promise((resolve, reject) => {
   if(correct){
       const cheese = {color: 'yellow', brand: null};
       resolve(cheese);
   } else {
       const rej = new Error('There is no cheese');
       reject(rej);
   }
});
let findCheese = () => {
   cheeseType
       .then(cheese => console.log('Cheese color' + cheese.color))
       .catch(e => console.log(e.message));
};
findCheese();
```

Async Await functions are different from the promise notation as they remove the .then .catch syntax in return for a more user readable syntax for simple code. Instead of declaring a new promise, an async function would be declared, and an await would be added to the main body of the function that calls the async. Nested async functions do not lend to better readability, so in many cases the .then syntax is easier to use.

```
let correct = true;
async function cheeseType(){
   return new Promise((resolve, reject) =>{
       if(correct){
           const cheese = {color: 'yellow', brand: null};
```

```
            resolve(cheese);
        } else {
            const rej = new Error('There is no cheese');
            reject(rej);
        }
    });
}
let findCheese = async () => {
    try{
        let cheese = await cheeseType();
        console.log(cheese.color);
    } catch (e) {
        console.log(e.message);
    }
}
(async () => {
    await findCheese();
})();
```

In either case, whether using .then or async await, the functionality of the code remains the same, so it depends on the coder or style guide to choose the appropriate usage.