JavaScript Promises

Promises are a fundamental concept in modern Javascript, providing a way to ~~handel~~ handle asynchronus operations more elegantly and avoid the infamous "callback hell".

They represent a value that may not be available immediately but will be resolved at some point in future.

3 Promise States

1. Pending : initial state
2. fulfilled : success state
3. Rejected : Failed state

Making a Promise

```
const burgerPromise = newPromise ((resolve, reject) => {
  if (haveBuns && havePatties) {
    resolve ("Here's your Burger"); //result
  } else
    reject ("sorry, we're out of ingradients."); //error
  }
});
```

2 Key Methods of Promise

1. .then() : handles result
2. .catch() : handles error

Use Case :

```
burgerPromise
  .then (burger => eatBurger (burger))
  .catch (error => orderPizza (error));
```

result

Chaining Promises

One of the most powerful features of Promises is the ability to chain them:

```
getBun()
  .then(addPatty)
  .then(addCheese)
  .then(addLettuce)
  .then(addSauce)
  .catch(dropBurger)
```

```
// fetchData()
  .then(processData)
  .then(saveData)
  .then(notifyUser)
  .catch(~~error~~ showError)
```

Promise.all()

It is like ordering party pack.

```
Promise.all([burgerPromise, friesPromise, drinkPromise])
  .then(meall => enjoyMeal(meal))
  .catch(error => orderPizza(error));
```

You will get your meal will all' items are ready. But if ~~one~~ even one item is unavailable, you're calling Pizza.

Promise.race()

It is like sending your friends to different restaurants and eating what comes ~~first~~. back first.

```
Promise.race([burgerPromise, FriesPromise, PizzaPromise])
  .then(food => devour(food))
  .catch(error => eatSnacks(error));
```

Async/Await

It is like going through a drive-thru. It looks more straightforward, but it's still a Promise under the hood.

```
async function getDinner() {                    //fetchAndProcessData()
  try {
    const burger = await orderBurger();         //fetchData()
    const fries = await orderFries();           //processData(data)
    return [burger, fries];                     ⇓
                                                saveData
  } catch (error) {
    return orderPizza();
  }
}
```

Error Handling

Always be prepared for things to go wrong. It's like carrying some emergency snacks.

```
burgerPromise
  .then(burger => eatBurger(burger))
  .catch(error => {
      console.log("Burger joint messed up", error);
      return orderPizza();
  });
```