```
step1(function (value1) {
    step2(value1, function(value2) {
        step3(value2, function(value3) {
            step4(value3, function(value4) {
                // Do something with value4
            });
        });
    });
});
```

```
step1
.then(step2)
.then(step3)
.then(step4);
```

¿Qué es una promesa?

- Una promesa es un objecto que representa un valor que va a estar disponible en el futuro
- Una promesa se define como un objeto que tiene una función como valor para la propiedad, `then`
- La función tiene que retornar una nueva promesa

```
then(success, reject, progress)

doSomething()
.then(doOtherThing)
.then(null, console.log); // si falla la primera o la segunda promesa
```

- Cuando la promesa se resulve se llama al parámetro success
- Cuando la promesa se rechaza se llama al parámetro reject

Ejercicios (1, 2, 3)

Creación y resolución de promesas

Ejercicios (4, 5, 6, 7)

Uso de las promesas

Puedes devolver un valor o una promesa, si devolvemos un valor Q lo convierte a promesa

```
.then(function () {
    return 3;
})
```

¿Qué pasa si falla `doSomething`?

¿Qué pasa si falla `doOtherThing`?

¿Y si falla la función que maneja el error?

```
doSomething()
.then(doOtherThing, function() {
    console.log("error");
});
```

```
doSomething()
.then(doOtherThing, function() {
    console.log("error");
})
.done();
```

Convertir funciones sincronas en asincronas

```
Q.fcall(function() {
    return 2;
})
.then(doOtherThing)
.then(doOtherThing)
.then(doOtherThing)
.done();
```

Esperar a que se completen varias promesas

```
Q.all([promesa1, promesa2])
.then(function (result) {
    console.log(result[0]); //resultado promesa1
    console.log(result[1]); //resultado promesa2
})
```

```
Q.all([promesa1, promesa2])
.spread(function (result1, result1) {

})
```

```
return getUsername()
.then(function (username) {
    return [username, getUser(username)];
})
.spread(function (username, user) {

});
```

```
function authenticate() {
    return getUsername()
    .then(function (username) {
        return getUser(username);
    })
    .then(function (user) {
        return getPassword()
        .then(function (password) {
            if (user.passwordHash !== hash(password)) {
                throw new Error("Can't authenticate");
            }
        });
    });
}
```

- promise.thenResolve(value)
- promise.thenReject(reason)
- promise.timeout(ms, message)
- promise.get(propertyName)
- promise.post(methodName, args)
- promise.delay(ms)
- promise.isPending()
- etc

Ejercicios (8, 9)

Bonus