Web Services

- Send/Receive data from server
- Without page navigation
    - HTML remains
    - JS state remains
    - can react to data received

Asynchronous (async)

Using Web services requires understanding async behavior

We already do async with event handlers!

- Code in a callback to happen later

Key theme: You must be async yourself to respond to async event

Pyramid of Doom

```
callsACallback(
  callsACallback(
    callsACallback(
      callsACallback(
        finalReaction();
      )
    )
  )
);
```

Nested callbacks (nested async handling) is hard to understand, maintain.

Promises

A **Promise** object is

- pending
- resolved (fulfilled)
- rejected

You can attach callbacks:

- For if/when resolved
- For if/when rejected

When it hits that state

- Callbacks are placed on event queue

Promises Details

Callbacks are still callbacks

- still called in reaction to event
- event is promise fulfillment

```
Promise.resolve() // returns promise in resolved state
.then( () => {
  console.log('finished');
});
```

Simple promise example

```
console.log(1);
returnsAPromise().then( () => console.log(2) );
console.log(3);
```

always logs `1 3 2`. **Always**

Why?

Chaining

```
const one = Promise.resolve();
const two = one.then( () => console.log(1) );
const three = two.then( () => console.log(2) );
```

VS

```
Promise.resolve()
  .then( () => console.log(1) )
  .then( () => console.log(2) );
```

Chained example

```
returnsAPromise()
  .then( () => console.log(1) )
  .then( () => console.log(2) );
```

Always `1 2`. **Always**. Why?

Resolve values

Promises might "resolve" with a value

- This value is passed to any callbacks
- This is **NOT** returned by the `then()` call

```
const promise = Promise.resolve("hi");
const value1 = promise.then(
  (text) => console.log(`callback: ${text}`)
);
console.log(`from then: ${value1}`);
```

```
from then: [object Promise]
callback: hi
```

Remember: `then()` returns a new promise

Golden rule: To use a value from async, you must stay async

Resolve with what

- A promise resolves with a value
- `.then()` on a promise returns a new promise

What value does the new promise resolve with?

- The return value of the callback
- If that return value is a promise
    - uses resolution of THAT promise

Chaining returns

When a callback returns a value

- Becomes the resolve value of promise of that `then()`

```
const result = Promise.resolve(1)
  .then( val => {
    console.log(val);
    return val+1;
  })
  .then( val => {
    console.log(val);
    return val+1;
  })
  .then( val => {
    console.log(val);
    return val+1;
  });
```

What is `result`?

Trick question!

```
const result = Promise.resolve(1)
  .then( val => {
    console.log(val);
    return val+1;
  })
  .then( val => {
    console.log(val);
    return val+1;
  })
  .then( val => {
    console.log(val);
    return val+1;
  });
```

`result` is a PROMISE

- that resolved with value 4
- but `result` is NOT 4

Promise resolve 1

```
const result = Promise.resolve(4)
  .then( (val) => val+1 );
result.then( val => console.log(val) );
```

What is the output?

Promise resolve 2

```
const result = Promise.resolve(4)
  .then( (val) => val+1 )
  .then( () => 2 )
  .then( (val) => val+3 );
result.then( val => console.log(val) );
```

What is the output?

Promise resolve 3

```
const result = Promise.resolve(4)
  .then( (val) => val+1 )
  .then( () => Promise.resolve(2) );
result.then( val => console.log(val) );
```

What is the output?

Promise resolve 4

```
const result = Promise.resolve(1)
  .then( (val) => val+1 )
  .then( () => Promise.resolve(4) )
  .then( (val) => Promise.resolve(val+4) );
```

What is the output?

Try/Catch is useless with Promises!

```
try {
  Promise.resolve()
    .then( () => {
      console.log(1);
      throw new Error("poop");
    });
} catch(err) {
  // Doesn't happen
  console.log(`caught ${err}`);
}
console.log(2);
```

Why? (Hint: output is `2 1`)

catch()

Promises `catch` method covers "failures"

- any thrown errors INSIDE a promise
- any returned **rejected** (vs **resolved** or **pending**) Promises

```
Promise.resolve()
  .then( () => {
    throw new Error("poop");
  })
  .then( () => console.log('does not happen') )
  .catch( err => console.log(err) );
```

`catch()` also returns a promise - **resolved** by default!

- Allows you to handle errors and keep going

Async/Await

A newer syntax is `async` and `await`

- A different way to manage promises
- Hides the `.then()` and `.catch()`
- Implicitly sets all following code to be async
- Allows try/catch

Do not use async/await for this course

Until you know promises very comfortably, async/await can cause confusion by hiding what is really happening

Once out of this class, feel free to use async/await