# JavaScipt Tips: Asynchronous Programming

# Loading Collections of Triangles

```
import { OBJLoader } from "../libs/CS559-Three/examples/jsm/loaders/OBJLoader.js";

let loader = new OBJLoader();
loader.load("./objects/07-astronaut.obj",
            function(astronaut) {
                astronaut.position.set(1.5, 4, 0);
                astronaut.scale.set(0.5, 0.5, 0.5);
                scene.add(astronaut);
                renderer.render(scene, camera);
            }
        );
```

Deferred loading

# JavaScript Tip of the Day: Asynchronous Programming

Loading might take time... don't wait!

```
loader.load("file.obj");
```

What to do when the object is ready?

- Callbacks
- Promises
- Asynchronous Functions

# Connecting to what we know

Web Browser programming uses an **event** model

- do "short duration" things in response to events
- don't block - return to browser
- schedule "special events" for future things
    - animation loops
    - break up big computations

# Asynchronous: Callbacks

```
import { OBJLoader } from "../libs/CS559-Three/examples/jsm/loaders/OBJLoader.js";

let loader = new OBJLoader();
loader.load("./objects/07-astronaut.obj",
            function(astronaut) {
                astronaut.position.set(1.5, 4, 0);
                astronaut.scale.set(0.5, 0.5, 0.5);
                scene.add(astronaut);
                renderer.render(scene, camera);
            }
        );
```

Give a **function** to call when things are ready

Effectively an **event handler** - we return to browser

# Asynchronous: Callbacks

```
loader.load("./objects/07-astronaut.obj",
            function(astronaut) {
                astronaut.position.set(1.5, 4, 0);
                astronaut.scale.set(0.5, 0.5, 0.5);
                scene.add(astronaut);
                renderer.render(scene, camera);
            }
        );
/** Code after the load is "started" **/
```

1. Code "after load start" is run immediately
2. Code "after load start" has no access to the object
3. Code "after load start" returns to browser
4. Code "after load ends" is in the callback function

# Some Notes on Asynchronicity

Still the "event" model

- loading finished is an "event"
- when it happens it gets put in the queue
- your function must return to the browser so it can process queue

Other approaches are "syntactic sugar" to make this nicer.

# Asynchronous: Promises

```
let obj = loader.loadAsync("./objects/07-astronaut.obj");
obj.then(function(astronaut) {
    astronaut.position.set(-2, 4, 0);
    astronaut.scale.set(0.5, 0.5, 0.5);
    scene.add(astronaut);
    renderer.render(scene, camera);
    });
```

1. the "loader object" is a **promise** - not an `Object3D`
2. this is just a different syntax for the callback

# Asynchronous Functions

```
let astro = await loader.loadAsync("./objects/07-astronaut.obj");
console.log(astro);
astro.position.set(-0, 4, -2);
astro.scale.set(0.5, 0.5, 0.5);
scene.add(astro);
renderer.render(scene, camera);
```

Only works inside of **asynchronous functions**

Program as if your are waiting for the result - other things continue

Really doing a promise - actually does return to browser

(but its hidden)

# then vs. await

```
loadAsync("thing1").then(
    /** do stuff with thing 1 */
    );
loadAsync("thing2").then(
    /** do stuff with thing 2 */
    );
loadAsync("thing3").then(
    /** do stuff with thing 3 */
    );
```

Starts 3 loads immediately

- parallel web fetches!

Finishes in any order

```
let o1 = await loadAsync("thing1");
/** do stuff with thing 1*/

let o2 = await loadAsync("thing2");
/** do stuff with thing 2*/

let o3 = await loadAsync("thing3");
/** do stuff with thing 3*/
```

Starts load 2 after 1 finishes

Forces the order

Object 1 will be loaded first

Object 1 is ready for Object 2

# Error Handling

Provide **two** functions

- what happens if the load succeeds
- what happens if the load fails

`Promise.then` takes two functions as arguments

Error handling is optional - but a good idea

# Some notes on Async in THREE

- Object loaders (usually) provide `Group` not `Mesh`
- Image loaders provide placeholders
  - you can use the image before its ready
  - you just see a temporary thing
- Class framework code provides placeholders