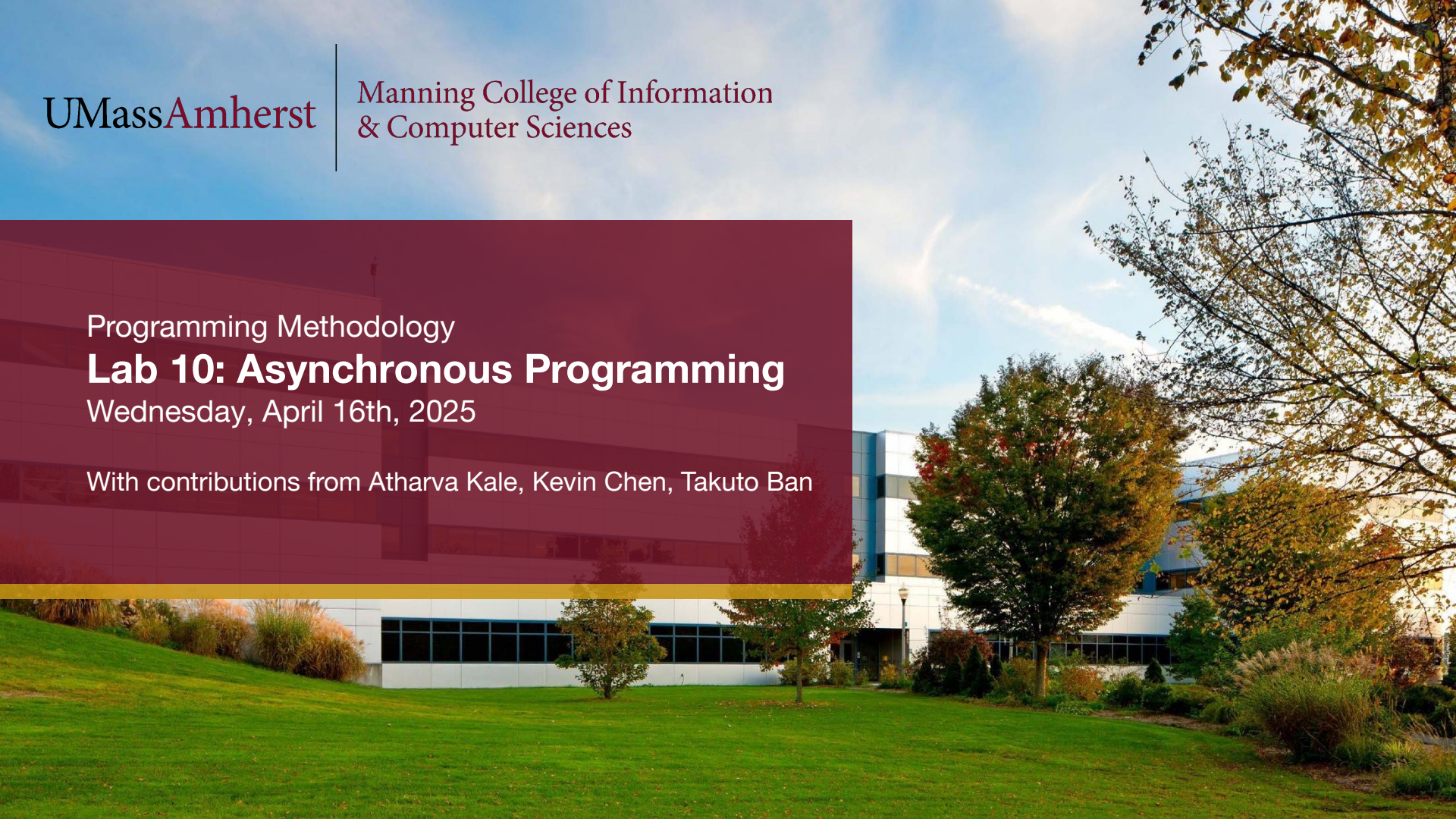


UMassAmherst | Manning College of Information & Computer Sciences

Programming Methodology

Lab 10: Asynchronous Programming

Wednesday, April 16th, 2025

With contributions from Atharva Kale, Kevin Chen, Takuto Ban

Agenda

Weekly Reminders

Motivate Async Programming

then

Exercise 1
Get hands-on with Promise API

Exercise 2
Scavenger Hunt

Use Gradescope

Reminders

Get in touch with your team for HW8!

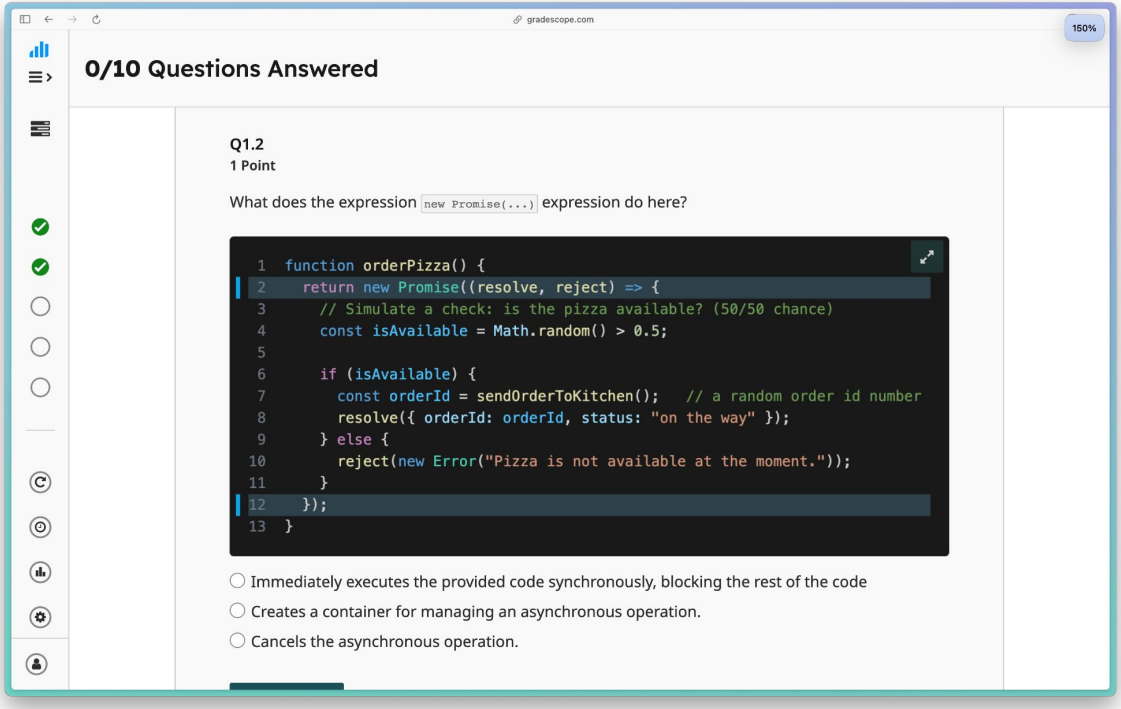
Join the Github repo

Email mkuechen@umass.edu for team issues

HW8 due 4/30
In two weeks!

New Lab Format for Today

We have some questions on Gradescope today :)

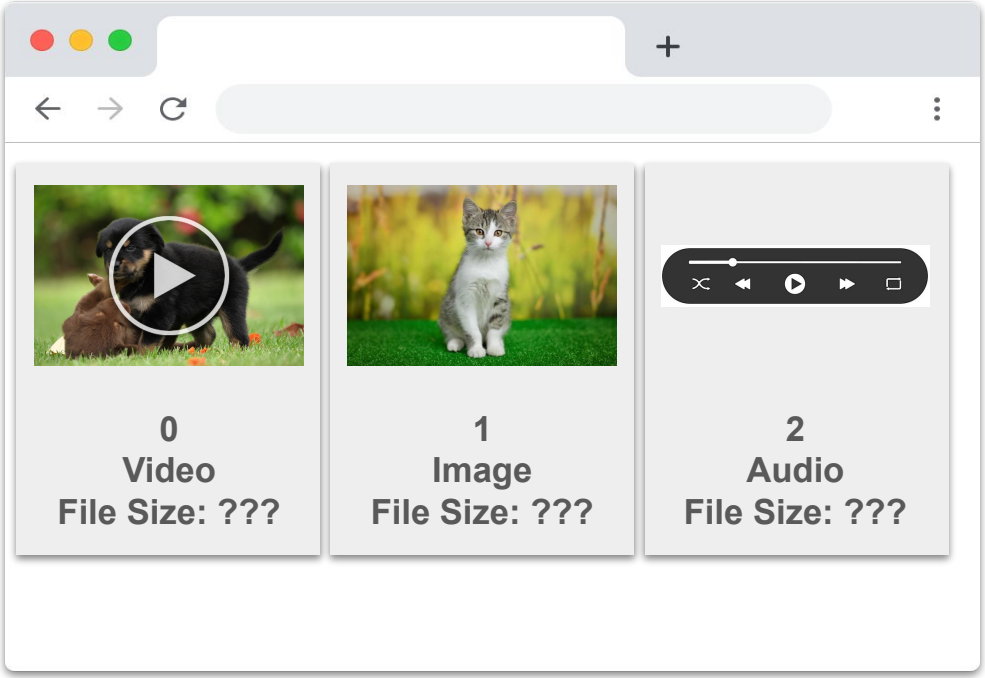


Motivation - Asynchronous Programming

Let's say you are developing a browser responsible for displaying three items:

1) A video
2) An image
3) An audio

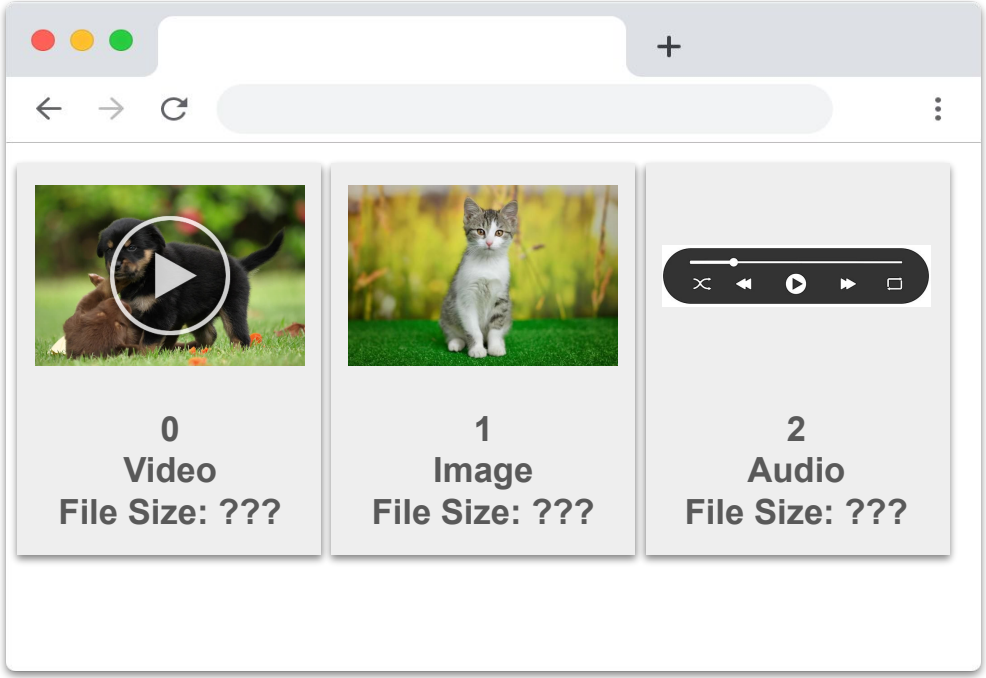
Goal: Give user best user experience



Motivation - Asynchronous Programming

We have three URLs:

```
1  const urls = [
2    "http://catsanddogs.com/video.mp4",
3    "http://catsanddogs.com/image.png",
4    "http://catsanddogs.com/audio.mp3",
5  ]
```



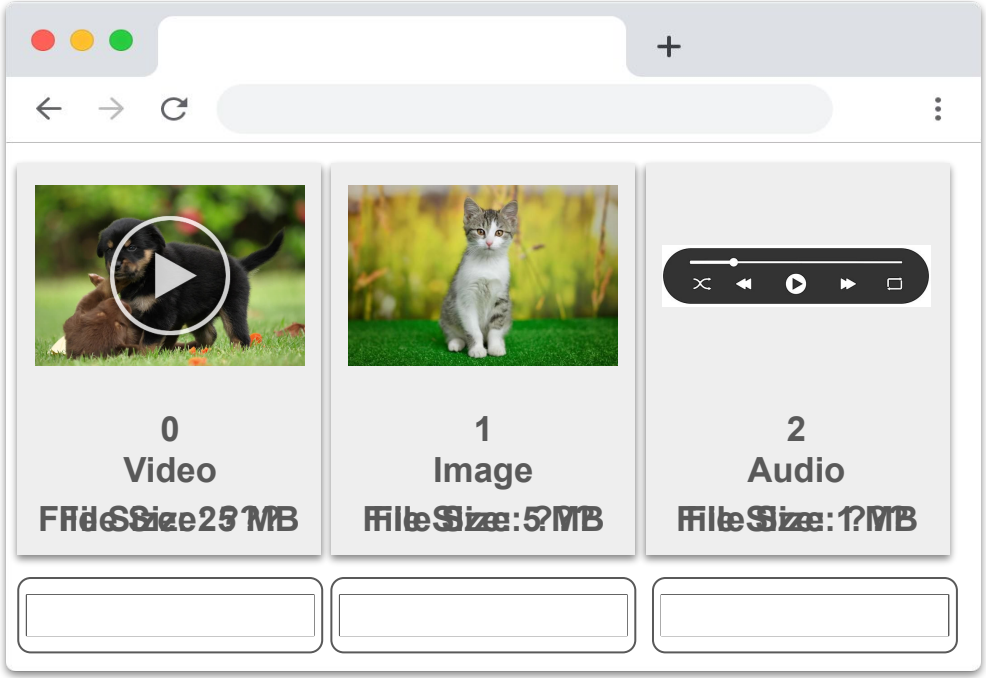
Motivation - Asynchronous Programming

With a basic implementation, we could:

1) Download all assets sequentially and then display them all at once to the user

or....

2) Download an asset and display it sequentially



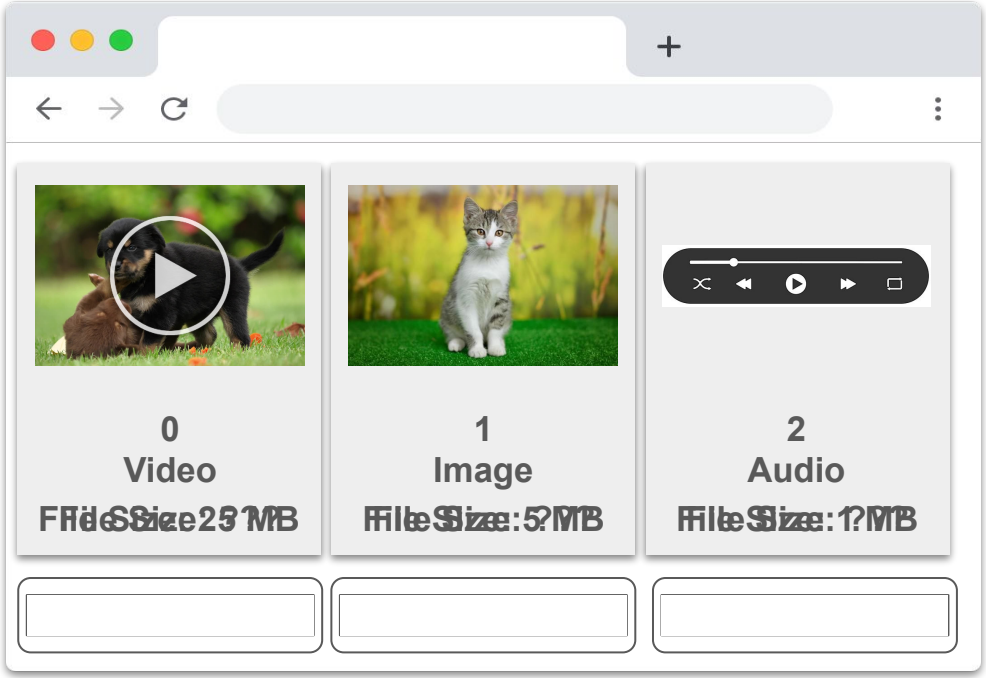
Motivation - Asynchronous Programming

But what if we could start downloading them all at once

AND

Display each right after they finish downloading?

The Promise API lets us do that!



Creating Promises

```
1  // main.js
2
3  const p1 = new Promise((resolve) => {
4    // <an operation that takes 10 secs to complete>
5    console.log("p1 completed");
6  });
7
8  const p2 = new Promise((resolve) => {
9    // <an operation that takes 5 secs to complete>
10   console.log("p2 completed");
11 });
12
13 console.log("main.js can continue work here");
14
```

```
1  // Console
2
3  main.js can continue work here
4  p2 completed
5  p1 completed
```

Resolving Promises

```
1  // Promises can "resolve" to a value
2
3  const p1 = new Promise((resolve) => {
4    console.log("p1 completed");
5    resolve("p1 resolved");
6  });
7
```

What can we do with the fulfillment value?

Resolving Promises

```
// We can chain promises using `.then`

const p1 = new Promise((resolve) => {
  console.log("p1 completed");
  // NOTE: resolve is a function that
  // accepts the fulfilled value
  resolve("p1 resolved");
})

const p2 = p1.then(
  // NOTE: fulfilledValue is the SAME
  // as the value we resolved with
  (fulfilledValue) => {
    console.log(fulfilledValue);
  },
);
// NOTE: p2 is ALSO a promise
```

```
// Console

p1 completed
p1 resolved
```

Read the MDN docs for the full specification of `Promise.prototype.then`

Promises that “reject”

```
// Promises can also "reject" with an error

const p1 = new Promise((resolve, reject) => {
  console.log("p1 completed");
  reject("p1 rejected");
});

const p2 = p1.then(
  (fulfilledValue) => {
    // NEVER runs
    console.log("fulfilledValue:", fulfilledValue);
  }
).catch((rejectedValue) => {
  // THIS WILL RUN
  console.log("rejectedValue:", rejectedValue);
});
```

```
// Console

p1 completed
rejectedValue: p1 rejected
```

Read the MDN docs for the full specification of `Promise.prototype.catch`

Promise API

Awesome!
We have all the building blocks to get started!

Let’s analyze some asynchronous code!

Exercise 1: Promises

Analyze the behavior of asynchronous code:

- **Get the starter code** from the website
- Open Exercise 1 in ex1.js
- In your groups, discuss the behavior of the code
- Answer the questions on **Gradescope**

TAs will walk around to help ensure you are on the right track

UMassAmherst | Manning College of Information & Computer Sciences

Combining Promises

Task: Write a function `countSucc` that **accepts an array of promises** and returns a Promise that fulfills with the number of successful promises from the input array

```
10  // countSucc: (Promise<T>[]) => Promise<number>
11  function countSucc(promiseArr) { ... }
```

Definition from the docs:

`Promise.allSettled(promiseArr)`: returns a Promise that always fulfills with an array of objects describing the outcome of each input promise:

```
{ status: "fulfilled" | "rejected", value?: someType, reason?: errType }
```

Combining Promises

Task: Write a function `countSucc` that **accepts an array of promises** and returns a Promise that fulfills with the number of successful promises from the input array

```
10  // countSucc: (Promise<T>[]) => Promise<number>
11  function countSucc(promiseArr) {
12    return Promise.allSettled(promiseArr)
13      .then(results => results.reduce((acc, e) => {
14        if (e.status == 'fulfilled') return acc + 1
15        else return acc
16      }, 0))
17  }
```

Definition from the docs:
`Promise.allSettled(promiseArr)`:
returns a Promise that always fulfills with an array of objects describing the outcome of each input promise:
`{ status: "fulfilled" | "rejected", value?: someType, reason?: errType }`

Exercise 2: Promises

Scavenger Hunt!

- You are tasked with the **finding a password** hidden across campus. Time is limited!
- Use Promise API to search for password pieces in a number of locations asynchronously
- Assemble the password to *save the day!*

Exercise 2: Promises

UMassAmherst | Manning College of Information & Computer Sciences

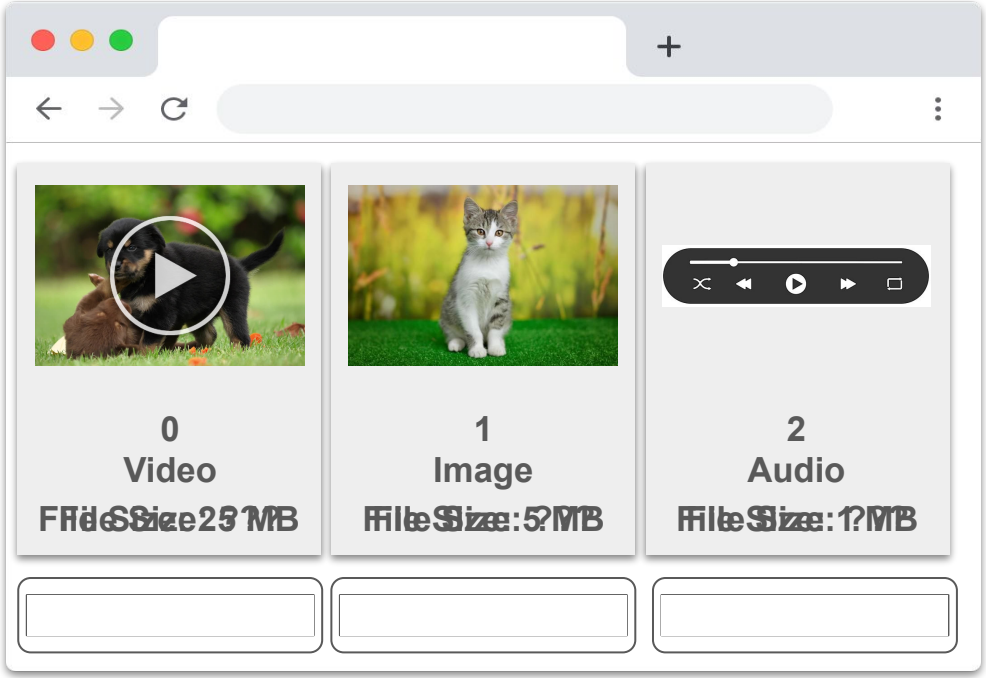
Scavenger Hunt!


```
38  const promise = searchClueAtLocation("LGRT")
39  // returns Promise<{ part: "geese", index: 3 }>
```

Basic Promise API

We know how to:

- Create Promises
- Chain Promises
- Handle rejected Promises
- But how do we **start several asynchronous tasks** at once and then **combine the result**?



Exercise 2: Promises

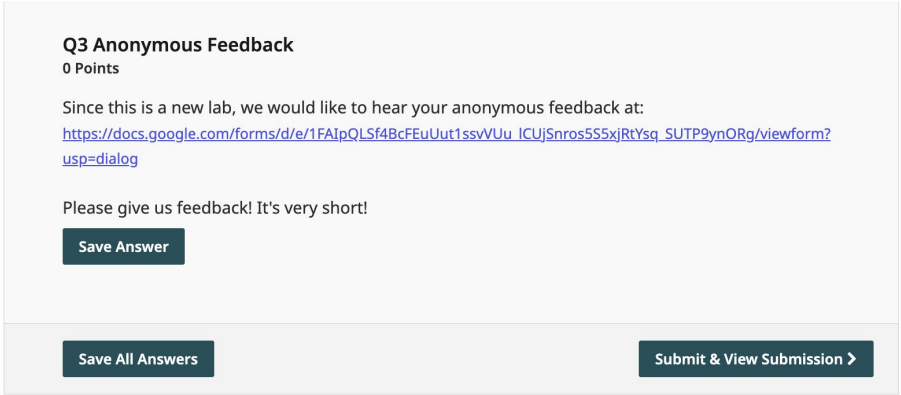
Scavenger Hunt!

- **Get the starter code** from the website
- Open Exercise 2 in ex2.js
- In your groups, **find the password**
- Answer the questions on Gradescope

TAs will walk around to help ensure you are on the right track

Exercise 2: Promises

Lab Feedback



Anonymous Feedback

Exercise 2: Promises

Scavenger Hunt Solution!

```
const passwordParts = Array(locations.length).fill("")
const searchPromises = locations.map(searchClueAtLocation);
Promise.allSettled(searchPromises).then(results => {
    results.forEach(result => {
        if (result.status === "fulfilled") {
            const { part, index } = result.value
            passwordParts[index] = part
        }
    });
    const password = passwordParts.join("");
    console.log(`Password: ${password}`);
    completeHunt(password);
});
```