Promise.all()

It's likely the case that you'll run into an API here or there that has paginated data. To ensure that you don't wind up awaiting every single element inside of a loop (which could take quite a while depending on the speed of your connection/API request), you can make the calls inside of a map, and await them with a `Promise.all()`.

If we take the code from the previous module, we can add to it a list of URLs. The subreddit `r/birdswitharms` is great, and regardless of whether or not someone visiting our site wants to see it, we want to show it to them. Now, instead of making multiple calls explicitly, we can keep all of our URLs in an array and map over them like so:

```
...

const urls = [`r/${props.subreddit}.json`, `r/birdswitharms.json`];

const subredditArrays = urls.map(async url => {
  const response = await fetch(url);
  const json = await response.json();
  const allPosts = json.data.children.map(obj => obj.data);

  return allPosts;
});

console.log(subredditArrays)

...
```

Awaiting the Promised Data:

Curiously, if we just did this, we would not wind up with a response we could use. Notice that when you `console.log(subredditArrays)`, what returns is an array of `[Promise], [Promise]`. Here's where we want to use `Promise.all`. Because we have to separate network calls going out, we want to wait until we get all of the data back:

```
...

const urls = [`r/${props.subreddit}.json`, `r/birdswitharms.json`];

const subredditArrays = urls.map(async url => {
  const response = await fetch(url);
  const json = await response.json();
  const allPosts = json.data.children.map(obj => obj.data);

  return allPosts;
});

const awaitedThings = await Promise.all(subredditArrays);

console.log("THINGS", awaitedThings);

...
```

Flattening Data:

Now, we have our awaited things! However, because each of our network calls is responding with an array, we now have a multidimensional array of response items. Our code is written entirely for a single dimensional array, so let's flatten that array, and set our state:

```
...

const urls = [`r/${props.subreddit}.json`, `r/birdswitharms.json`];

const subredditArrays = urls.map(async url => {
  const response = await fetch(url);
  const json = await response.json();
  const allPosts = json.data.children.map(obj => obj.data);

  return allPosts;
});

const awaitedThings = await Promise.all(subredditArrays);

console.log("THINGS", awaitedThings);
const allPosts = awaitedThings.flat();

console.log("All posts: ", allPosts);
myCache.set(cacheKey, allPosts, 100);
setAllPosts(allPosts);
setPosts(allPosts);

...
```

Randomizing:

Our data are returning just fine now, however, it looks as if all of our data are entirely in blocks of `[r/aww], [r/birdswitharms]`. To mix the data up, we can bring in `lodash/shuffle`:

```
...

const urls = [`r/${props.subreddit}.json`, `r/birdswitharms.json`];

const subredditArrays = urls.map(async url => {
  const response = await fetch(url);
  const json = await response.json();
  const allPosts = json.data.children.map(obj => obj.data);

  return allPosts;
});

const awaitedThings = await Promise.all(subredditArrays);

console.log("THINGS", awaitedThings);
const allPosts = awaitedThings.flat();

console.log("All posts: ", allPosts);
const shuffledData = shuffle(allPosts);

myCache.set(cacheKey, shuffledData, 100);

setAllPosts(shuffledData);
setPosts(shuffledData);

...
```