ENSF 381
Full Stack Web Development

Lecture 17:
Asynchronous JavaScript

Slides: Ahmad Abdellatif, PhD

Instructor: Novarun Deb, PhD

UNIVERSITY OF CALGARY

Outline

- Introduction to API and related terminologies.
- Asynchronous programming.
- Common errors in fetching data.

What is Application Programming Interface (API)?

- A set of rules and protocols for building software applications that allow them to communicate with each other over the internet.

- It defines the methods and data formats that applications can use to request and exchange information.

- Web APIs are commonly used to enable the integration of different software systems, allowing them to work together and share data.

API Terminologies

- **Endpoint**: An API endpoint is a specific URL or URI that an API exposes for interacting with its resources.

- **HTTP Methods**: APIs use standard HTTP methods to perform operations on resources. Common methods include GET (retrieve data), POST (create data), PUT/PATCH (update data), and DELETE (remove data).

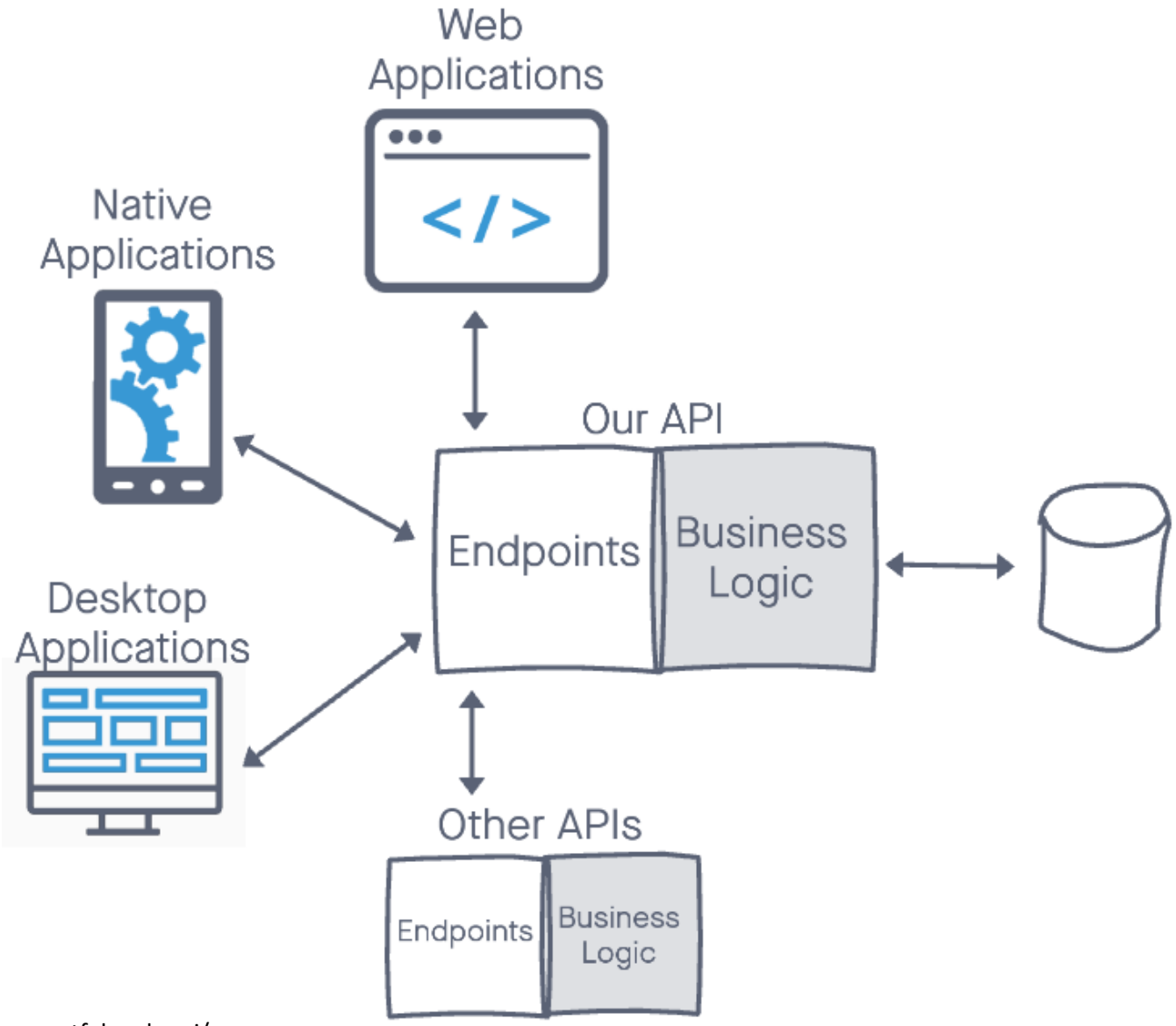
- **Request and Response**: Communication with a web API involves making HTTP requests to specific endpoints. The API then processes the request and returns a response, usually in JSON or XML format.

API Terminologies

- **Authentication**: Many APIs require authentication to ensure that only authorized users or applications can access the data or perform specific actions.

- **Documentation**: API providers often provide documentation that explains how to use the API, including details about available endpoints, request formats, response formats, and any authentication requirements.

Example on API



https://www.dotnetnakama.com/blog/designing-a-restful-web-api/

We can do much using APIs…

- YouTube API: allows to display videos on a website.
- Weather API: provides an access to current weather data, hourly, 5- and 16-day forecasts.
- Flight API: retrieves live flight price tracking, airport schedule, and live flight tracking.
- Financial Modeling Prep API: provides access to a wide range of financial data, including live and historical stock prices.

Tasks take some time to complete

- There are tasks often have to wait for some work to finish before they can be completed:
  - Fetch data from external servers.
  - Access a database.
  - Stream video or audio content.

So…

Performing time-consuming operations can introduce delays that would negatively impact the user experience.

1. These tasks could cause the application to wait until they are completed before moving on to the next task such as reading files.

2. JavaScript does not wait for tasks to be completed; it continues executing code. The subsequent lines of code will be executed before the tasks are complete, resulting in incomplete results such as fetching data from the server.

Asynchronous programming

- A programming paradigm in which the execution of code does not occur in a sequential and blocking manner.

- Asynchronous programming allows certain operations to be initiated and continue executing without waiting for their completion.

- Asynchronous tasks in JavaScript do not impede the main thread.

JavaScript Promises

- Promises are a mechanism for handling asynchronous operations in a more structured and manageable way.

- An objects used to represent the eventual completion or failure of an asynchronous operation and its resulting value.

- You can think of a promise as:

“I Promise a Result! Either way, I’ll come back and let you know how it went”

Async/Await syntax

```
async function fetchData() {
  try {
    //make an asynchronous request to the specified URL
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');

    //Parse the response body as JSON; wait for this asynchronous operation to complete
    const data = await response.json();

    // Process the fetched data
    console.log('Fetched Data:', data);
  } catch (error) {
    // Handle errors
    console.error('Error fetching data:', error);
  }}
```

`async:` declares a function as asynchronous that returns a Promise.

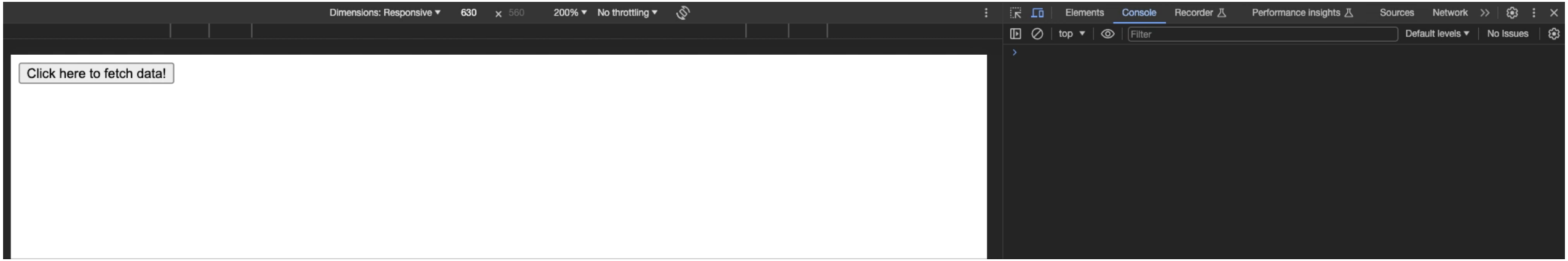
`fetch(URL):` A function used to make network requests, typically to retrieve data from a server.

`await:` within an asynchronous function, you can use the await keyword to pause the execution of the function until the Promise is resolved.

Async/Await - example

```html
<!DOCTYPE html>
<html><head>
  <title>Fetch Data Example</title>
</head>
<body>
<button onclick="fetchData()">Click here to fetch data!</button>
<script>
// Function to fetch data asynchronously using the Fetch API
async function fetchData() {
  try {
    const response = await fetch('https://jsonplaceholder.typicode.com/todos/1');
    const data = await response.json();
    // Process the fetched data
    console.log('Fetched Data:', data);
  } catch (error) {
    // Handle errors
    console.error('Error fetching data:', error);
  }
}
</script>
</body>
</html>
```

Async/Await - example



Populate HTML elements using third-party API

GET https://api.randomuser.me/?nat=US&results=1 Send

Params • Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (16) Test Results Status: 200 OK Time: 381 ms Size: 1.87 KB Save as example

Pretty Raw Preview Visualize JSON


```
{
    "results": [
        {
            "gender": "female",
            "name": {
                "title": "Miss",
                "first": "Carla",
                "last": "Wood"
            },
            "location": {
                "street": {
                    "number": 608,
                    "name": "Smokey Ln"
                },
                "city": "Mcallen",
                "state": "South Carolina",
                "country": "United States",
                "postcode": 69142,
                "coordinates": {
                    "latitude": "-48.6027",
                    "longitude": "49.4637"
                },
                "timezone": {
                    "offset": "+6:00",
                    "description": "Almaty, Dhaka, Colombo"
                }
            },
            "email": "carla.wood@example.com",
            "login": {
                "uuid": "2f0c0c8c-a9fa-49a3-aece-cf2f1c77ceb9",
                "username": "tinyrabbit966",
                "password": "shirley",
```

Populate HTML elements using third-party API

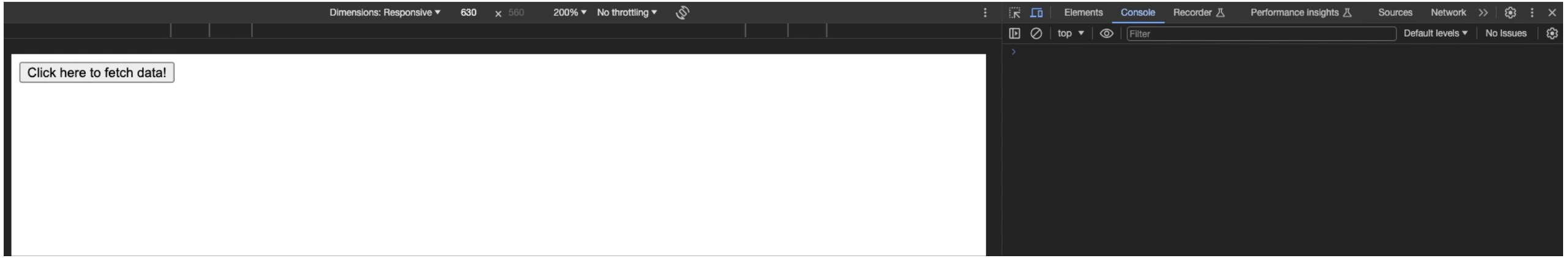
```
<!DOCTYPE html>
<html>
<head>
  <title>Fetch Data Example</title>
</head>
<body>

<button onclick="populateData()">Click here to fetch data!</button>
<label id="email-label"></label>
<label id="cell-label"></label>

<script>
  async function populateData(){
  try{
      let res = await fetch("https://api.randomuser.me/?nat=US&results=1");
      let {results} = await res.json();
      let{email,cell} = results[0]

      document.getElementById("email-label").textContent = email
      document.getElementById("cell-label").textContent = cell
  }
  catch (error) {console.log(error)}
  }
</script></body></html>
```

Populate HTML elements using third-party API



What is the output of this code?

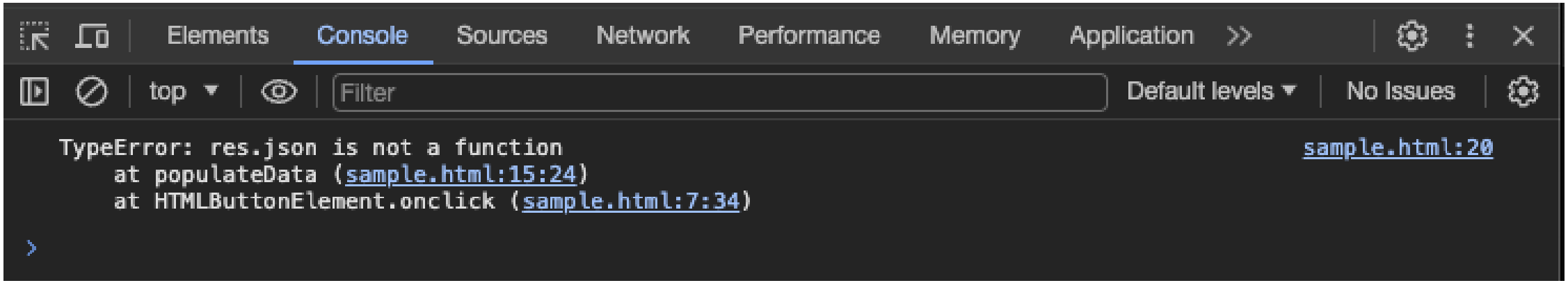
```
<!DOCTYPE html>
<html>
<head>
  <title>Fetch Data Example</title>
</head>
<body>

<button onclick="populateData()">Click here to fetch data!</button>
<label id="email-label"></label>
<label id="cell-label"></label>

<script>
  async function populateData(){
  try{
      let res =       fetch("https://api.randomuser.me/?nat=US&results=1");
      let {results} =       res.json();
      let{email,cell} = results[0]

      document.getElementById("email-label").textContent = email
      document.getElementById("cell-label").textContent = cell
  }
  catch (error) {console.log(error)}
  }
</script></body></html>
```

What is the output of this code?



Interactive data display: Fetching and rendering on the fly


```
[
    {
        "id": 1,
        "name": "Leanne Graham",
        "username": "Bret",
        "email": "Sincere@april.biz",
        "address": {
            "street": "Kulas Light",
            "suite": "Apt. 556",
            "city": "Gwenborough",
            "zipcode": "92998-3874",
            "geo": {
                "lat": "-37.3159",
                "lng": "81.1496"
            }
        },
        "phone": "1-770-736-8031 x56442",
        "website": "hildegard.org",
        "company": {
            "name": "Romaguera-Crona",
            "catchPhrase": "Multi-layered client-server neural-net",
            "bs": "harness real-time e-markets"
        }
    },
```

Interactive data display: Fetching and rendering on the fly

```html
<!DOCTYPE html>
<html lang="en">
<head> <title>Fetch Data Example</title> </head>
<body>
    <div id="data-container"></div>
    <script>
        async function fetchData() {
            try {
                const response = await fetch('https://jsonplaceholder.typicode.com/users');
                const users = await response.json();

                const dataContainer = document.getElementById('data-container');
                users.forEach((user) => {
                    const userDiv = document.createElement('div');

                    const nameElement = document.createElement('h2');
                    nameElement.textContent = user.name;

                    const emailElement = document.createElement('p');
                    emailElement.textContent = `Email: ${user.email}`;

                    const phoneElement = document.createElement('p');
                    phoneElement.textContent = `Phone: ${user.phone}`;

                    userDiv.appendChild(nameElement);
                    userDiv.appendChild(emailElement);
                    userDiv.appendChild(phoneElement);

                    dataContainer.appendChild(userDiv);
                });
            }
```

Interactive data display: Fetching and rendering on the fly

```
    catch (error) {
                console.error('Error fetching data:', error);
    }
   }

      // Call the async function
      fetchData();

    </script>
  </body>
</html>
```

Interactive data display: Fetching and rendering on the fly

file:///Users/ahmadabdellatif/Desktop/ENSF381/fetch.html

Leanne Graham

Email: Sincere@april.biz

Phone: 1-770-736-8031 x56442

Ervin Howell

Email: Shanna@melissa.tv

Phone: 010-692-6593 x09125

Clementine Bauch

Email: Nathan@yesenia.net

Phone: 1-463-123-4447

Patricia Lebsack

Email: Julianne.OConner@kory.org

Phone: 493-170-9623 x156

Chelsey Dietrich

Email: Lucio_Hettinger@annie.ca

Phone: (254)954-1289

Mrs. Dennis Schulist

Email: Karley_Dach@jasper.info

Phone: 1-477-935-8478 x6430

Kurtis Weissnat

Email: Telly.Hoeger@billy.biz

Phone: 210.067.6132

Nicholas Runolfsdottir V

Email: Sherwood@rosamond.me

Phone: 586.493.6943 x140

Glenna Reichert

Common errors in fetching data

- When fetching data from a server using JavaScript, several common errors can occur:
  - **No Internet Connection:** the user's device is not connected to the internet, the fetch request will fail.
  - **Server Unreachable:** The server might be down, or there could be an issue with the server's domain.
  - **Incorrect URL:** A typo or an incorrect endpoint can result in a failed request.
  - **Server-Side Errors:** the server may encounter an internal error (e.g., 500 Internal Server Error) when processing the request.

Common errors in fetching data

- **Invalid JSON Format:** the server returns data in an unexpected format or if the response is not valid JSON when expected, parsing the response as JSON could fail.

- **Timeouts:** the server takes too long to respond, a timeout error may occur. This could happen due to heavy server load or network issues.

Questions