Introduction to JSON

JavaScript Object Notion or JSON

- Data interchange format
- Human and machine-readable
- Text-based format

Why use JSON?

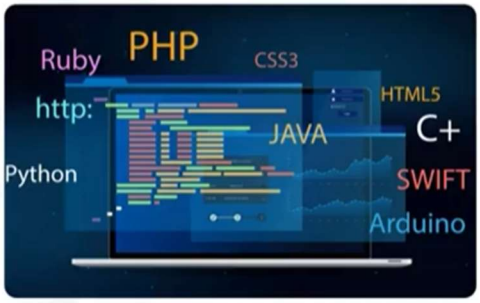
- Data interchange:
  - Server and web or different software system parts
  - Common communication language
  - Example: API request is in JSON format

Why use JSON?

- Human-readable:
  - Assist in reading and writing code
  - Use simple syntax
  - Used for debugging, configuration, editing

Why use JSON?

- Language agnostic:
  - Not tied to programming language
  - Used in any programming environment
  - Supported through libraries and APIs



Why use JSON?

- Simplicity:
  - Simpler that XML
  - No complex tags and attributes

How is JSON different from objects?

Aspect	JSON	Object
Text vs. data structures	• A text-based data interchange format. • Represents data as a string. • Primarily used for data exchange or storage.	• Fundamental data structures in programming. • Hold data and methods to manipulate it. • Model entities or concepts in software.
Syntax	• Use key-value pairs and is limited to predefined data types. • Keys are enclosed in double quotes.	• Not restricted to a specific syntax or set of data types. • Can include a variety of data types.
Purpose	• Used for data interchange. • Facilitate data transfer and interoperability. • Storing configuration data.	• Model and manipulate data within the application. • Represent entities or concepts in the program's logic.

JSON vs. Object: Example

JSON (in JavaScript):

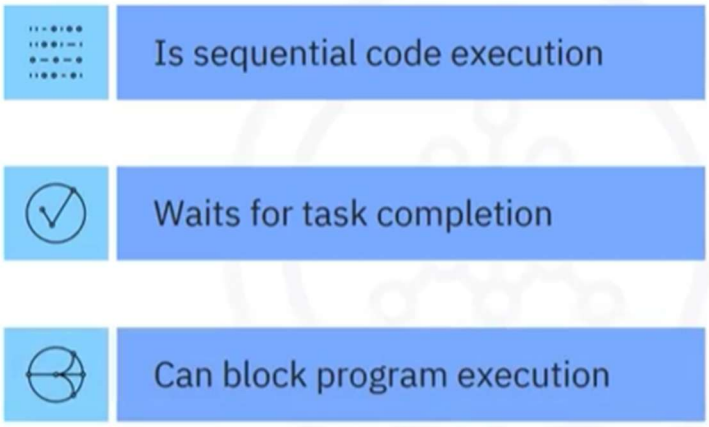
```
{
  "name": "John Doe",
  "age": 30,
  "city": "New York"
}
```

Object (in JavaScript):

```
const person = {
  name: "John Doe,"
  age: 30,
  city: "New York,"
  sayHello: function() {
    console.log("Hello, my name is "
+ this.name);
  }
};
```

Introduction to Async and Sync Execution

Synchronous programming



Synchronous code: Example 1

```
console.log("Task 1");
console.log("Task 2");
console.log("Task 3");
```

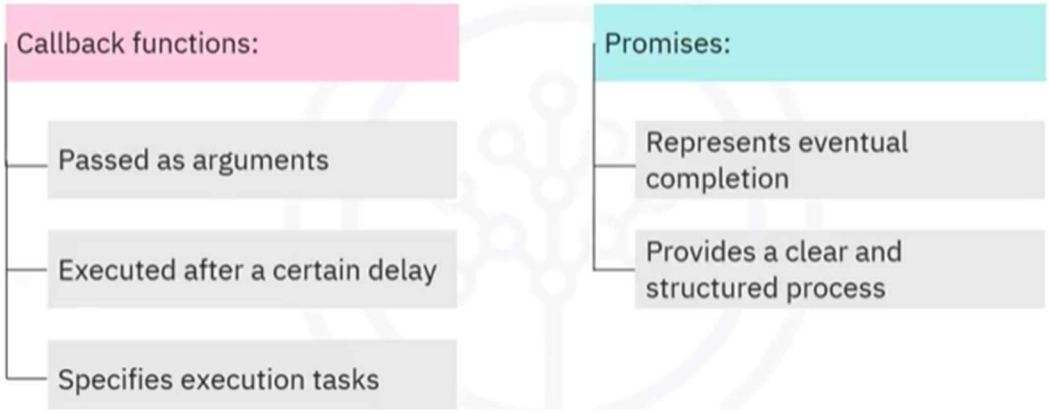
Synchronous code: Example 2

```
function greet(name) {
  console.log("Hello, " + name + "!");
}
function askQuestion() {
  console.log("How are you today?");
}
function farewell(name) {
  console.log("Goodbye, " + name + "!");
}
console.log("Start of the program");
greet("Alice");
askQuestion();
farewell("Bob");
console.log("End of the program");
```

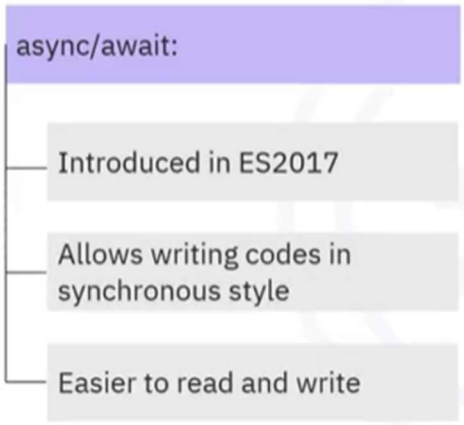
Asynchronous programming

- Executes code without blocking the execution thread
- Enables tasks to run concurrently
- Handles time-consuming tasks
- Examples: Network requests, file I/O, user interactions

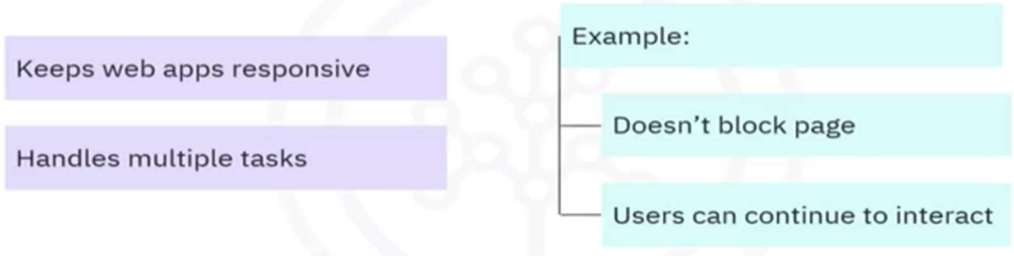
Asynchronous programming: Key concepts



Asynchronous programming: Key concepts



Asynchronous programming: Importance



Sync vs. Async programming

Feature	Synchronous	Asynchronous
Order of execution	Code is executed in order, one step at a time, blocking the main thread	Code doesn't wait; it can continue executing other operations while asynchronous tasks are in progress
Blocking vs. Non-blocking	Blocks the program's execution while a task is in progress, potentially making it unresponsive	Doesn't block the program; it allows other tasks to run alongside the asynchronous one, improving responsiveness
Callbacks vs. Continuation	Relies on the natural order of code execution; each line follows the previous one	Uses callbacks, Promises, or async/await to manage and control the flow of asynchronous operations
Use cases	Suitable for simple scripts, mathematical calculations, or tasks that don’t involve waiting for external resources.	Ideal for handling I/O operations like network requests, file reading/writing, and tasks that require waiting for external events or user interactions