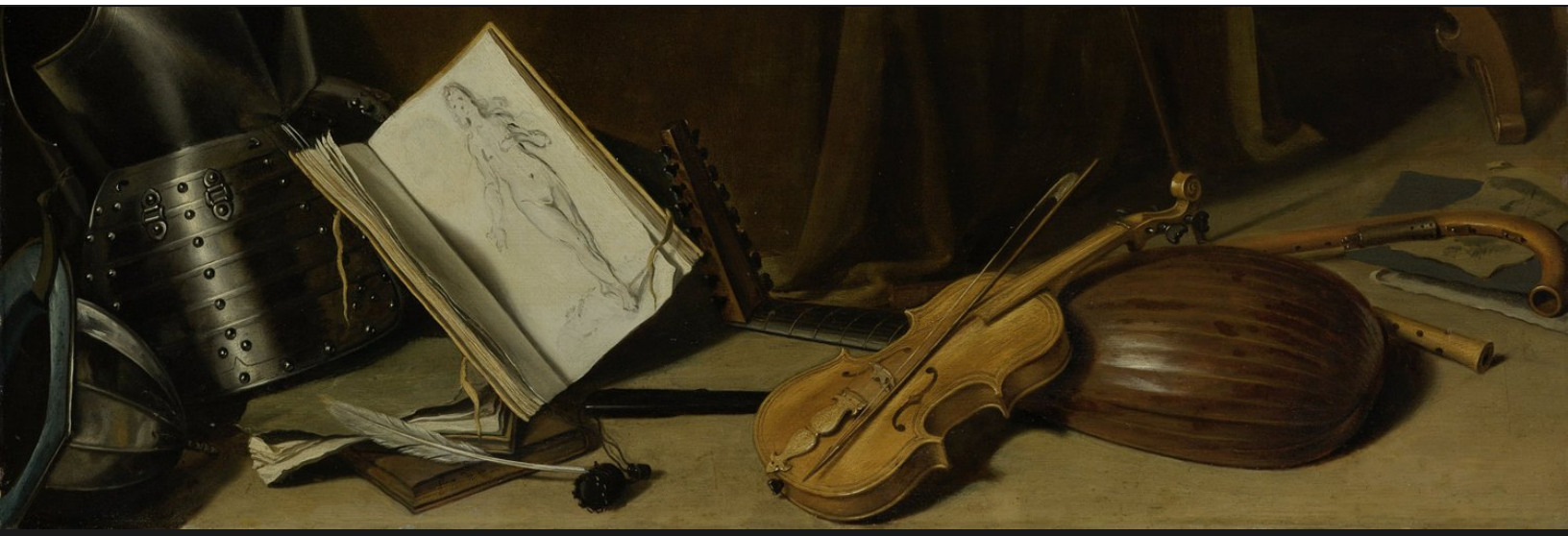

Week 1.2

Created	December 27, 2023 8:02 PM
Created by	A Aditya Kulkarni
Tags	Empty

Week 1.2: JavaScript Foundation

Why Languages?

Computers operate using machine code, a binary system that is difficult for humans to understand and work with directly. Programming languages serve as a bridge between human-readable code and machine code, enabling programmers to communicate instructions to computers effectively.

Key Components of a Computer:

- **Computer:** The hardware that executes instructions.
- **RAM (Random Access Memory):** Temporary storage for data that the CPU can quickly access.
- **SSD (Solid State Drive):** Non-volatile storage for long-term data storage.

Purpose of Languages:

- **Abstraction:** Provide a higher level of abstraction for human understanding.
- **Efficiency:** Enable efficient communication with the computer.

Interpreted vs Compiled Languages

Interpreted Languages:

- Code is executed line by line.
- Requires an interpreter at runtime.
- Examples: Python, JavaScript.

Compiled Languages:

- Code is translated into machine code before execution.
- Results in a standalone executable file.
- Examples: C, C++, Java.

JavaScript vs Other Languages in Some Use-Cases

JavaScript Advantages:

- **Versatility:** Can be used for both client-side (browser) and server-side (Node.js) development.
- **Ecosystem:** Extensive libraries and frameworks (e.g., React, Angular).
- **Dynamic Typing:** Allows flexibility in variable types.

Strict vs Dynamic Languages

Strict Languages:

- **Static Typing:** Type of a variable is known at compile time.
- **Examples:** Java, C++.

Dynamic Languages:

- **Dynamic Typing:** Type of a variable is determined at runtime.
- **Examples:** JavaScript, Python.

Single-Threaded Nature of JavaScript

- JavaScript is single-threaded, meaning it has one execution thread.
- Uses an event-driven, non-blocking I/O model for asynchronous operations.
- Example:

```
console.log("Start"); setTimeout(() => { console.log("Inside setTimeou
t"); }, 1000); console.log("End");
```

Simple Primitives in JavaScript

- **Number, Strings, Booleans:**

```
let num = 42; let str = "Hello, World!"; let bool = true;
```

Complex Primitives in JavaScript

- **Arrays, Objects:**

```
let arr = [1, 2, 3]; let obj = { key: "value", num: 42 };
```

Functions in JavaScript

- **Function Declaration:**

```
function add(a, b) { return a + b; }
```

- **Function Expression:**

```
const multiply = function (a, b) { return a * b; };
```

Practice Problem Solving

- Solving coding challenges to enhance problem-solving skills.

Callback Functions, Event Loop, Callback Queue

- **Callback Function:**

```
function fetchData(callback) { setTimeout(() => { callback("Data receive
d"); }, 1000); } fetchData((data) => { console.log(data); });
```

- **Event Loop:**
  - Manages the execution of the call stack and callback queue.
- **Callback Queue:**
  - Queue that holds callback functions waiting to be executed.

Callback Hell and Promises

- **Callback Hell (Callback Pyramid):**
  - Nested structure of callback functions leading to code complexity.

```
fetchData((data) => { processData(data, (result) => { displayResult(resul
t, () => { // More nested callbacks... }); }); });
```

- Promises:
  - Handles asynchronous operations more elegantly.

```
const fetchDataPromise = () => { return new Promise((resolve) => { setTim
eout(() => { resolve("Data received"); }, 1000); }); }; fetchDataPromise
() .then((data) => processData(data)) .then((result) => displayResult(res
ult)) .catch((error) => console.error(error));
```

Assignments

1. Create a Counter in JavaScript

```
let count = 30; const countdown = setInterval(() => { console.log(count); cou
nt--; if (count < 0) { clearInterval(countdown); console.log("Countdown compl
ete!"); } }, 1000);
```

2. Calculate Time between `setTimeout` Call and Execution

```
console.log("Start"); const start = Date.now(); setTimeout(() => { const end
= Date.now(); const elapsedTime = end - start; console.log(`Time elapsed: ${e
lapsedTime} ms`); }, 1000); console.log("End");
```

3. Create a Terminal Clock (HH:MM:SS)

```
function formatTime(num) { return num < 10 ? `0${num}` : `${num}`; } functio
n displayTime() { const now = new Date(); const hours = formatTime(now.getHou
rs()); const minutes = formatTime(now.getMinutes()); const seconds = formatTi
me(now.getSeconds()); console.log(`${hours}:${minutes}:${seconds}`); } setInt
erval(displayTime, 1000);
```