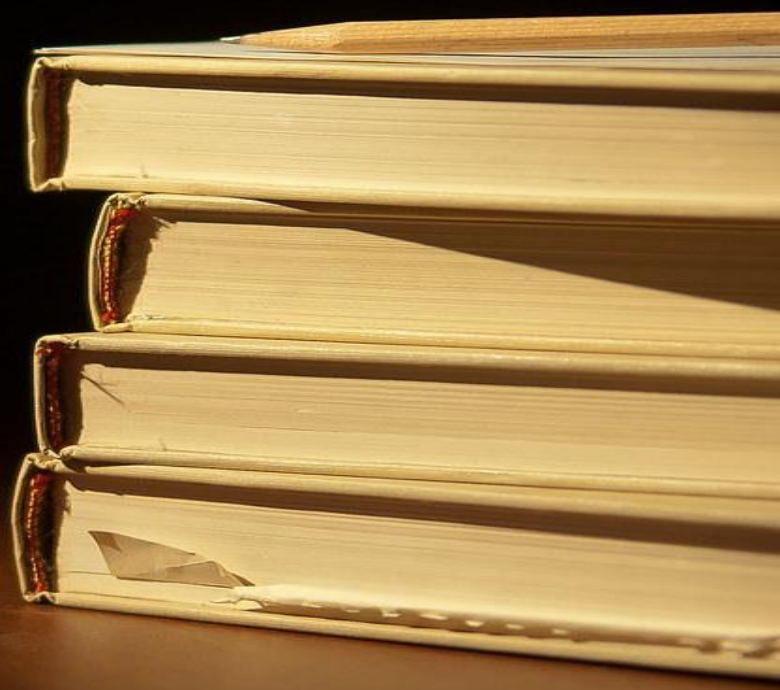
Welcome to lecture 4!



Agenda

Session Objectives

- Review homework
  - Cover the takeaways from the task
- Understand Scope & Hoisting
- Introduction to Arrays
  - What are methods?
  - What are arrow functions?
- Learn about Objects & JSON
  - Managing custom objects
  - Explore JSON
  - How does JSON help us in the client-server architecture?
- Q&A
- Homework (due next Saturday by 5PM)

Homework

Task: Shopping Cart Total Calculator

- Write a program that simulates a simple shopping cart calculator. Your program should:
  - Repeatedly prompt the user to enter the price of an item
  - If the user enters a negative number or a non-number value, display an alert saying "Invalid price, try again!" and skip adding that input
  - Continue prompting until the user clicks cancel
  - Calculate the total cost of the entered items
  - After the user finishes entering prices, use an if statement to check the total:
    - If the total is greater than 100, alert "Your total is Rs X. You qualify for a discount!"
    - Otherwise, alert "Your total is Rs X."
  - Write a function named calculateTotal that handles the price-collecting loop and returns the total

Solution Link (Remember: this isn't the only way to solve this problem. Your code may be different & can still be correct)

Takeaways

- We will learn concepts one at a time, but solving a problem requires many concepts
  - You learned to write a conditional statement inside a while loop inside a function!
- Using loops effectively
  - We learned to prompt a user continuously until they cancel
  - Avoiding overuse of if-else conditions using continue and break
- Input validation
  - Checking for negative numbers
  - Using isNan - a function that tells us if the argument is not a number
- Encapsulation for re-usability and better code organization
  - By writing a function, calculateTotal, we isolate the logic for collecting and validating input

Hoisting & Scope

Introduction to Scope

- What does the word 'scope' mean in everyday language?
  - Scope of a project
- Definition of scope
  - Scope defines where a variable or function is accessible
- Types of Scope
  - Global Scope
  - Local Scope (Function & Block level)

```
let globalVar = "Accessible Everywhere"; // Global scope

function checkScope() {
  let localVar = "Accessible Only Here"; // Local scope
  console.log(globalVar); // Accessible
  console.log(localVar);  // Accessible
}

checkScope();
console.log(localVar); // Error: not accessible here
```

Examples: Scope

- Global vs Local

```
var a = 1; // Global

function localTest() {
  var b = 2;
  console.log(a + b); // 3 (accessible)
}
console.log(b); // Error! b is local to function
```

- Block Scope

```
if (true) {
  const blockVar = "Inside block";
}
console.log(blockVar); // Output: ReferenceError: blockVar is not defined. Why? blockVar is block scoped
```

Examples: Scope

- Nested Scope

```
const greeting = "Hello"; // Global variable

function outerFunction() {
  let name = "Alice"; // Local to outerFunction
  console.log(greeting + ", " + name); // Both variables are accessible here

  function innerFunction() {
    let exclamation = "!"; // Local to innerFunction
    // Inner function can access variables from outerFunction and the global scope
    console.log(greeting + ", " + name + exclamation);
  }
  innerFunction();
}

outerFunction();
```

Introduction to Hoisting

- Can we use variables or functions before declaring them? — YES
- What is a hoisting?
  - A default behavior of Javascript where it moves declarations (not initializations) to the top of their scope before execution
  - Remember: A variable is declared using let/const/var. Initialization is when you give it a value.
- What gets hoisted by Javascript?
  - Function Declarations: Both the function's name and its body are hoisted, so you can call the function before its declaration in the code
  - Variable Declarations with var: The declaration is hoisted, but the assignment remains in place. This means you might see an undefined value if you try to use the variable before it is assigned
  - Variables with let and const: These are hoisted but are not initialized, which means accessing them before their declaration leads to an error (this period is known as the Temporal Dead Zone)
    - What's a Temporal Dead Zone?
      - A period in your code during which a variable declared with let or const exists in the scope but is not yet initialized
      - Purpose: helps prevent bugs by ensuring that variables are not accessed before they are explicitly declared and initialized, leading to more predictable and safer code

Examples: Hoisting

- Function hoisting

```
sayHello(); // This works because the function declaration is hoisted

function sayHello() {
  console.log("Hello, world!");
}
```

- Var hoisting

```
console.log(myVar); // Outputs: undefined (declaration hoisted, but not the assignment)
var myVar = 10;
console.log(myVar); // Outputs: 10
```

Examples: Hoisting

- Let hoisting

```
// Uncommenting this line will produce an error: ReferenceError: myLet is not defined
// Reason: a ReferenceError due to the Temporal Dead Zone.
// console.log(myLet);

let myLet = 20;
console.log(myLet); // Outputs: 20
```

- Const hoisting

```
console.log(c); // ReferenceError: c is not defined
const c = 10;
```

Arrays

What are Arrays?

- Beyond software engineering - what are arrays?
  - Array of flowers at a flower shop
  - Array of solar panels on the roof
- So, what are arrays in computer science?
  - Arrays store ordered collections of data in a single variable
  - Useful to manage related data items efficiently
  - The data can be of any data type: number, string, boolean, null, undefined. We'll learn more!
  - Arrays are non-primitive data types, aka, objects
- Fundamentals operations on an array
  - Access data: data is accessed based on an *index*
  - Mutable: arrays can be modified. You may add, remove or change data in an array
  - Heterogeneous: an array can store a mixture of any data types
  - Dynamic: arrays can grow or shrink their size as you mutate the data

```
// Declare an empty array
let emptyArray = [];
// Initialize an array with some elements
let fruits = ["apple", "banana", "orange"]
console.log(fruits); // Outputs: [ 'apple', 'banana', 'orange' ];
```

Examples: Arrays

Accessing an index:

```
// Access the first element (index 0)
console.log("First fruit:", fruits[0]); // Outputs: apple

// Access the last element using the length property
console.log("Last fruit:", fruits[fruits.length - 1]); // Outputs: orange
```

Iterating through an array:

```
// Using a for loop to iterate over the array
for (let i = 0; i < fruits.length; i++) {
  console.log("Fruit at index", i + ":", fruits[i]);
}
```

Updating an index in an array:

```
let fruits = ["apple", "banana", "orange"]
fruits[0] = "guava";
console.log(fruits); // output: ["guava", "banana", "orange"]
```

Array Methods

What are methods?

- Generally speaking, how do you define a method?
  - A method to perform an experiment
  - Methods in arts: “method acting”
  - Method to staying organized
- Methods in programming
  - A method is a function that is a property of an object
  - In the context of arrays, array methods are built-in functions that help you interact with a variable
- Method vs Function
  - A function is an independent series of statements that performs a task
  - A method is a function that is associated with an object and cannot be called independently

```
let fruits = ["apple", "banana", "orange"]
let fruit = fruits.at(2); // retrieves value at index 2; similar to fruits[2]
fruits.push("mango"); // adds a new element to the array
const removedFruit = fruits.pop();     // removes last element in the array ("mango") and returns it
```

Array Methods: Examples

shift() vs unshift()

```
let arr = [1, 2, 3];
let first = arr.shift();
console.log("After shift(), removed:", first); // Outputs: 1
console.log("Array now:", arr);                // Outputs: [2, 3]

arr.unshift(0);
console.log("After unshift(0):", arr);         // Outputs: [0, 2, 3]
```

concat(): merges two or more arrays into a new array without changing the original

```
const num1 = [1, 2, 3];
const num2 = [4, 5, 6];
const num3 = [7, 8, 9];
const numbers = num1.concat(num2, num3);
console.log(numbers); // [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Arrow Functions

- What are arrow functions?
  - Arrow functions provide a concise syntax for writing functions
  - Instead of using the *function* keyword, you use the *arrow* (=>) notation
  - Particularly useful when writing short functions that doesn't need to be reused

```
// Traditional function expression
let double = function(x) {
  return x * 2;
};

// Arrow function equivalent
let doubleArrow = (x) => x * 2;
console.log(doubleArrow(5)); // Outputs: 10

// Multi-line arrow arrow function
let doubleAndWriteToConsole = (x) => {
  const doubled = x * 2;
  console.log(doubled);
  return doubled;
};
```

Examples: Arrow function using array

forEach: executes a provided function once for each element in the array:

```
let numbers = [1, 2, 3, 4, 5];

// Using forEach with an arrow function to print each number
numbers.forEach((num) => {
  console.log("Number:", num);
});
```

Map: creates a new array populated with results of called a provided function on each element of the array:

```
// Using map to create a new array where each number is doubled
let doubled = numbers.map((num) => num * 2);
console.log("Doubled:", doubled); // Outputs: [2, 4, 6, 8, 10]
```

Objects & JSON

Let’s construct Objects!

- Beyond programming, how do you define an object?
  - Our possessions: phone, wallet/purse, pen/pencils, etc
  - Beyond us: buildings, vehicles, industrial machines, etc
- Objects in programming
  - Objects are collections of key-value pairs that allow you to model and store data
  - Keys are usually strings and values can be any valid data type (numbers, strings, arrays, functions, or even other objects)
- Fundamentals of an object
  - Creation: commonly creating using literal notation ({}) or by using `new Object()` syntax
  - Properties & Methods
    - Properties hold data (eg: name, age, location, etc)
    - Methods: are functions stored as properties (eg: a function to calculate a bill for delivery)
  - Access
    - Use dot notation (person.name) or bracket notation (person[“name”]) to access a property in an object
  - Mutability
    - Objects are mutable - you may add, change or remove properties after creation

Examples: Object

- Person object

```
// Create an object using literal notation
let person = {                          // The variable, person, has three properties (name, age, city & state)
  name: "Rupam",
  age: 25,
  city: "Imphal",
  state: "Manipur"
};

console.log(person);                    // Output: {"name":"Rupam","age":25,"city":"Imphal","state":"Manipur"}
console.log(person.name);               // Dot notation; Output: "Alice"
console.log(person["age"]);             // Bracket notation; Output: 25

person.name = "Rajkumar";
person.age = 26;
console.log(person);                    // Output: {"name":"Rajkumar","age":25,"city":"Imphal","state":"Manipur"}
```

Object Methods

- A simple calculator

```
let calculator = {
  // Syntax supported by Javascript since ES3, released in 1999
  add: function(x, y) {                    // A method to add two numbers using the function keyword
    return x + y;
  },
  // Syntax supported by Javascript since ES6, released in 2015
  subtract(x, y) {                         // A method to subtract two numbers using method definition shorthand
    return x - y;
  },
  // Also supported since ES6
  multiple: (x, y) => {                     // A method multiply using arrow function
    return x* y;
  }
};

console.log(calculator.add(5, 3));         // Outputs: 8
console.log(calculator.subtract(10, 4));   // Outputs: 6
```

Pre-defined Object Methods

- What are predefined Object methods?
  - Easily iterate over JavaScript objects using built-in methods
- Why use these methods?
  - Simplify object iteration
  - Access keys, values, or both conveniently
  - Very useful when dynamically handling data

```
const student = {
  name: "Arjun",
  grade: 10,
  subject: "Science"
};

// Get all keys
const keys = Object.keys(student);
console.log(keys); // ["name", "grade", "subject"]
let values = Object.values(student);
console.log(values); // ["Alice", 10, "Science"]
```

Re-discovering JSON

- What is JSON?
  - Javascript Object Notation
  - Lightweight format used for storing and transmitting data
  - JSON, similar to Objects, has key/value pairs, but has certain strict rules on format:
    - Keys must be strings in double quotes
    - Value may be a string, number, boolean, array, object or null
    - Each key/value pair must be separated by commas
    - JSON doesn’t permit comments
- Uses
  - JSON is ideal for data exchange between a client and server
  - Before transmitting data, we must perform serialization using JSON.stringify(...)
  - Once server returns a response in JSON, deserialization is performed using JSON.parse(...)

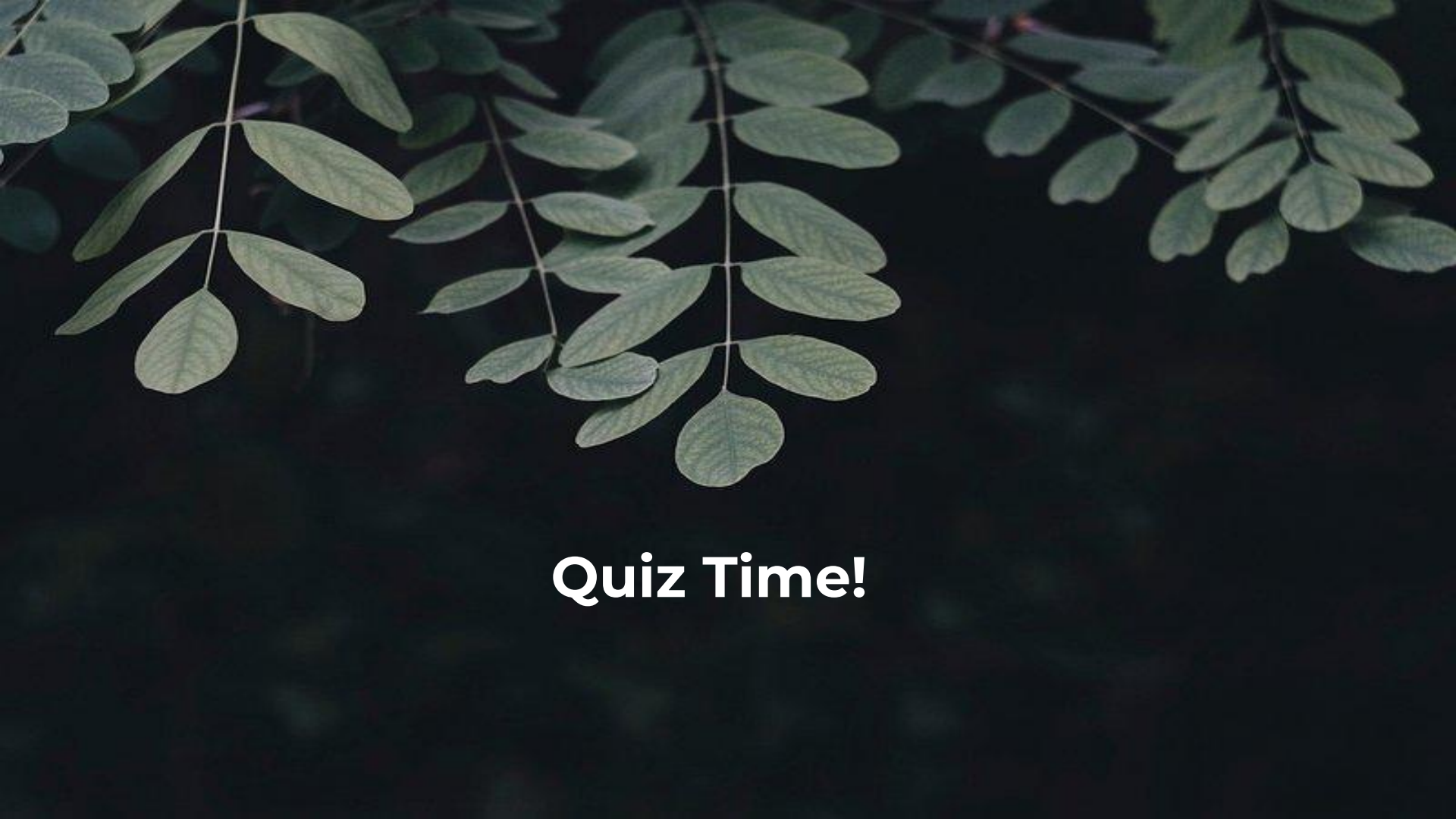
JSON Examples

- Serializing Object to JSON & Deserializing JSON to Object

```
let book = {
  title: "The Great Gatsby",
  author: "F. Scott Fitzgerald",
  year: 1925
};

let jsonString = JSON.stringify(book);                  // Serialization step
console.log(jsonString);                                // Outputs: '{"title":"The Great Gatsby","author":"F. Scott Fitzgerald","year":1925}'

// JSON representation of a user assigned to a variable ‘jsonData’
let jsonData = '{"name": "Charlie", "age": 25, "city": "New York"}';
let user = JSON.parse(jsonData);                        // Deserialization Step
console.log(user.name);                                 // Outputs: "Charlie"
console.log(user.age);                                  // Outputs: 25
```


Quiz Time!

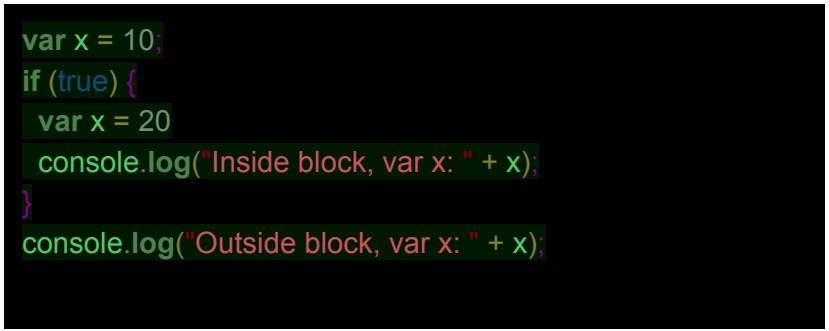
1. Hoisting & Scope

- What is the output of the code?
  - A) “Inside block, var x: 20” and “Outside block, var x: 20”
  - B) “Inside block, var x: 10” and “Outside block, var x: 20”
  - C) “Inside block, var x: 20” and “Outside block, var x: 10”
  - D) “Inside block, var x: 10” and “Outside block, var x: 10”

Correct Answer: A

Why?

- var x = 10 declares a global variable with value 10
- var x = 20 re-declares the global variable, now with a value of 20
- So, the value of x changes before console.log(“Inside block...”)



```
var x = 10;
if (true) {
  var x = 20
  console.log("Inside block, var x: " + x);
}
console.log("Outside block, var x: " + x);
```

2. Arrays & Methods

- What is the output of the code?
  - A) [10, 20, 30, 40]
  - B) [10, 20, 30, 5]
  - C) [10, 20, 30, 10]

```
let arr = [10, 20, 30];
arr.push(40);
arr.unshift(5);
arr.pop();
arr.shift();
arr.push(arr.at(0));
console.log(arr);
```

Correct Answer: C

Why?

- After push (adds new element) 40, arr: [10,20,30,40]
- After unshifting (adding to the beginning of the array) 5, arr: [5,10,20,30,40]
- After pop (removes last element), arr: [5,10,20,30]
- After shift (removes first element), arr: [10,20,30]
- After push, arr: [10,20,30,10] ← add an element, arr.at(0), to the end, which is the first element (10)

3. Objects & JSON

- How can I access name of employee “Alice?”
  - A) data[“employees”].0.name
  - B) data.employees.0.name
  - C) data.employees[0].name

```
let companyJSON = `{
  "company": "Tech Corp",
  "employees": [
    { "name": "Alice", "role": "Developer" },
    { "name": "Bob", "role": "Designer" }
  ]
}`;

let data = JSON.parse(companyJSON);
console.log(data.company);              // Outputs: "Tech Corp"
// Fill in the blank
console.log(___);
```

Correct Answer: C

Why?

- data is an object with a key, employees. So we access it using the dot notation
- But employees is an array, so we must use bracket notation to access an element inside the employees array
- employee[0] is an object, so we can use dot notation to access name

References

- Hoisting & Scope
  - W3C: Hoisting & Scope
  - Javascript Info (not beginner friendly): Scope & Hoisting using var
  - Mozilla Documentation: Hoisting & Scope
- Arrays & Methods
  - W3C: Arrays & Array Methods (recommended)
  - Mozilla Documentation: Arrays
  - Javascript Info: Arrays & Array Methods
- Objects & JSON
  - W3C: Objects & JSON
  - Mozilla Documentation: Objects & JSON
  - Javascript Info: Objects & JSON
- Functions vs Arrow Functions
  - Mozilla Documentation: Functions & Arrow Functions

Homework

- Link to homework exercises
- Please note that future homeworks, lectures, and all resources for each lecture will be in this Github repository