Verse of the day

Proverbs 26:14-15 "As a door turns on its hinges, so a **sluggard** turns on his bed. The **sluggard** buries his hand in the dish; he is too lazy to bring it back to his mouth."

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

Javascript Object

Outlines

1. Introduction to Javascript Objects
2. Creating Javascript Objects
3. Accessing and Modifying Javascript Object Properties
4. Javascript Object Methods
5. Javascript Object Inheritance
6. JSON

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

Javascript Object

1. Introduction to Javascript Object

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

1. Introduction to Javascript Object

- In JavaScript, an **object is a collection of properties**, where each property has a name and a value.
- A property's name is a string, and its value can be a string, number, boolean, function, or another object.
- Objects are often used to organize and structure code, since they allow you to group related data and functionality together.

1. Introduction to Javascript Object

- For example, let's say you're building an application to keep track of information about different cars.
  - You might create an object to represent each car, with properties such as make, model, year, and color.
  - The make and model properties would be strings, the year property would be a number, and the color property would be a string.
  - Additionally, you might include a method for starting the car.

1. Introduction to Javascript Object

- Here's an example of how you might create a JavaScript object to represent a car:

```
let car = {
    maker: "Toyota",
    model: "Camry",
    year: 2020,
    color: “Red",
    start: function() {
        console.log("The car is running.");
    }
};
```

1. Introduction to Javascript Object

You can also create an object using a constructor function, this is called as Object Constructor

```
function Car(make, model, year, color) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.color = color;
    this.start = function() {
        console.log("The " + this.make + " " + this.model + "
is running.");
    };
}
let myCar = new Car("Toyota", "Camry", 2020, "Blue");
```

1. Introduction to Javascript Object

- You can access the properties and methods of an object using dot notation or bracket notation. For example, to access the make of the car, you would use the following:

```
console.log(car.make); // Toyota
console.log(myCar["make"]); // Toyota
```

- And to call the start method

```
car.start(); // "The car is running."
myCar.start(); // "The Toyota Camry is running."
```

- In short, objects are a fundamental concept in JavaScript and are used to store and organize data, as well as to group related functionality together. As you continue to learn JavaScript, you'll find that objects are used in many different ways, such as in the DOM (Document Object Model) for interacting with HTML and CSS, and in many JavaScript libraries and frameworks.

2. Creating objects

In JavaScript, there are several ways to create objects. The two most common ways are:

1. Using object literals: An object literal is a way of creating an object by specifying its properties and values directly in a set of curly braces {}. The properties and values are separated by colons, and each property and value pair is separated by a comma. Here's an example of an object created using an object literal:

2. Creating objects

```
let person = {
    name: "John Doe",
    age: 30,
    occupation: "Developer"
};
```

This creates an object with three properties: "name", "age", and "occupation", with respective values "John Doe", 30, "Developer".

2. Creating objects

2. Using object constructors: An object constructor is a special function that is used to create objects. The keyword "this" within the constructor function refers to the new object being created. You can create an object by calling the constructor function with the keyword "new". Here's an example of how you might use an object constructor to create a Car object

```
function Car(make, model, year, color) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.color = color;
}

let myCar = new Car("Toyota", "Camry", 2020, "Blue");
```

Javascript Object

2. Creating Object

2. Creating objects

- Object constructors are generally preferred for creating multiple instances of the same object with different values, as opposed to object literals, which are more suitable for creating single instances of an object with a fixed set of properties and values.
- Also, you can create an object by using **Object.create(proto)** where proto is the object to use as the prototype of the newly created object. This method allows you to create an object with a specific prototype, which can be useful when you want to create an object that inherits from another object.

2. Creating objects

```
let personProto = {
    sayHi: function() {
        console.log("Hi! My name is " + this.name);
    }
}
let person = Object.create(personProto);
person.name = "John Doe";
person.sayHi(); // Hi! My name is John Doe
```

- In summary, you can create javascript object with object literals, object constructors and **Object.create(proto)** method. Each one of them has its own use cases, so you should choose the one that better fits your needs

Examples of Creating Objects

- **Example 1** Using object literals to create a book object:

```
let book = {
    title: "The Catcher in the Rye",
    author: "J.D. Salinger",
    pageCount: 214,
    isAvailable: true,
    summary: function() {
        console.log(this.title + " is written by " + this.author
+ " and has " + this.pageCount + " pages.");
    }
};
```

Examples of Creating objects

- **Example 1** Using object literals to create a book object:

```
console.log(book.title); // "The Catcher in the Rye"
console.log(book.isAvailable); // true
book.summary(); // "The Catcher in the Rye is written by J.D.
Salinger and has 214 pages."
```

- In this example, we create an object literal to represent a book, which has properties such as title, author, pageCount, and isAvailable. We also added a method called summary() which log a summary message to the console.

Examples of Creating objects

- **Example 2** Using object constructors to create multiple Employee objects:

```
function Employee(name, position, salary) {
    this.name = name;
    this.position = position;
    this.salary = salary;
}

let employee1 = new Employee("John Smith", "Manager", 75000);
let employee2 = new Employee("Jane Doe", "Developer", 65000);
let employee3 = new Employee("Bob Johnson", "Designer", 60000);
```

Examples of Creating objects

- **Example 2** Using object constructors to create multiple Employee objects:

```
console.log(employee1.name); // "John Smith"
console.log(employee2.position); // "Developer"
console.log(employee3.salary); // 60000
```

- In this example, we create an object constructor function called Employee that takes three arguments: name, position, and salary. We then create three Employee objects and set their properties accordingly.

Examples of Creating objects

- **Example 3** Creating object using object.create()

```
let personProto = {
    sayHi: function() {
        console.log("Hi! My name is " + this.name);
    }
}
let person = Object.create(personProto);
person.name = "John Doe";
person.sayHi(); // Hi! My name is John Doe
```

- In this example, we first create an object called **personProto** with a method called **sayHi**. We then use the **Object.create()** method to create a new object called person, which inherits from the **personProto** object. We set the name property on the person object and call the **sayHi** method, which logs a greeting to the console.

Javascript Object

3. Accessing and Modifying Properties

FACULTY OF INFORMATION TECHNOLOGY

Universitas Advent Indonesia

3. Accessing and Modifying Properties

- Once you've created an object, you'll often need to access and modify its properties. There are two ways to do this: using dot notation and using bracket notation.

1. Using dot notation: Dot notation is the most common way to access and modify object properties. It involves using the name of the property followed by a dot (.) and the property name. Here's an example of how you might use dot notation to access and modify properties of a car object:

```
let car = {
    make: "Toyota",
    model: "Camry",
    year: 2020,
    color: "Blue"
};
```

3. Accessing and Modifying Properties

```
console.log(car.make); // Toyota
car.make = "Honda";
console.log(car.make); // Honda
```

- In this example, we first use dot notation to access the "make" property of the car object and log its value to the console. We then use dot notation to change the value of the "make" property to "Honda".

3. Accessing and Modifying Properties

2. Using bracket notation: Bracket notation is an alternative way to access and modify object properties. It involves using the name of the property inside square brackets, like an array. Here's an example of how you might use bracket notation to access and modify properties of a car object:

```
let car = {
    maker: "Toyota",
    model: "Camry",
    year: 2020,
    color: "Blue"
};
console.log(car["make"]); // Toyota
car["make"] = "Honda";
console.log(car["make"]); // Honda
```

3. Accessing and Modifying Properties

Bracket notation is useful when the property name is stored in a variable, and in this way, you can access the property dinamically, for example:

```
let propertyName = "maker";
console.log(car[propertyName]); // Toyota
car[propertyName] = "Honda";
console.log(car[propertyName]); // Honda
```

- Both dot notation and bracket notation can be used to access and modify object properties, and the choice between them often comes down to personal preference. However, it is worth noting that there are some differences between the two:

3. Accessing and Modifying Properties

1. Dot notation cannot be used when the property name starts with a number or contains spaces, while bracket notation can handle any property name.
2. Dot notation is slightly faster than bracket notation, but this is generally not an important consideration unless you are working with a very large number of properties.

- In short, you can use dot notation or bracket notation to access and modify properties of an object in javascript, but there are some subtle differences that you should keep in mind when choosing which notation to use.
- Examples of accessing and modifying properties using dot notation and bracket notation:

3. Accessing and Modifying Properties

1. Accessing properties of a **student** object:

```
let student = {
    name: "John Smith",
    age: 25,
    course: "Computer Science",
    grade: 80
};

console.log(student.name); // "John Smith"
console.log(student.course); // "Computer Science"
console.log(student["grade"]); // 80
```

- In this example, we use dot notation to access the "name" and "course" properties of the student object, and bracket notation to access the "grade" property.

3. Accessing and Modifying Properties

2. Modifying properties of a **book** object:

```
let book = {
    title: "The Catcher in the Rye",
    author: "J.D. Salinger",
    pageCount: 214,
    isAvailable: true
};

book.isAvailable = false;
console.log(book.isAvailable); // false
book["pageCount"] = 230;
console.log(book["pageCount"]); // 230
```

- In this example, we use dot notation to change the value of the "isAvailable" property to false, and bracket notation to change the value of the "pageCount" property to 230.

3. Accessing and Modifying Properties

3. Adding new properties to an object:

```
let animal = {
    type: "dog",
    breed: "Golden Retriever"
};

animal.color = "Golden";
animal["weight"] = 25;
console.log(animal.color); // "Golden"
console.log(animal["weight"]); // 25
```

- In this example, we use dot notation and bracket notation to add new properties "color" and "weight" to the animal object.

3. Accessing and Modifying Properties

4. Dynamic property names using variables:

```
let person = {
    name: "John Doe",
    age: 30,
    occupation: "Developer"
};

let property = "name";
let value = "Jane Doe";
person[property] = value;
console.log(person.name); // "Jane Doe"
console.log(person["age"]); // 30
```

3. Accessing and Modifying Properties

- In this example, we first create a variable called "property" that contains the string "name". We then use bracket notation to access the property "name" of the person object and set it to "Jane Doe" using the value variable.
- As you can see, dot notation and bracket notation can be used to access and modify properties of an object, and both notations are interchangeable. The choice between the two often comes down to personal preference and the specific needs of your code

Javascript Object

4. Javascript Object Methods

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

4. Javascript Object Methods

- In JavaScript, objects can have methods, which are functions that are associated with an object. Methods allow you to perform actions on an object, such as retrieving or changing its properties, or performing some other operation.

Here's an example of an object that has a method:

```
let car = {
    make: "Toyota",
    model: "Camry",
    year: 2020,
    color: "Blue",
    start: function() {
        console.log("The car is running.");
    }
};
```

4. Javascript Object Methods

- In this example, the car object has a method called **start()** that logs a message to the console when it is called. To call this method, you would use the following code:

```
car.start(); // "The car is running."
```

- Methods can also accept arguments, which are used to pass values into the method. Here's an example:

```
let calculator = {
    sum: function(x, y) {
        console.log(x + y);
    }
};
calculator.sum(3, 5); // 8
```

4. Javascript Object Methods

- In this example, the calculator object has a method called **sum** that accepts two arguments: **x** and **y**. When the method is called with the arguments 3 and 5, it logs the sum of those numbers (8) to the console.
- It's also common to see method that make use of this keyword, which refer to the object that's calling the method.

```
let person = {
    name: "John Doe",
    age: 30,
    occupation: "Developer",
    introduce: function() {
        console.log("Hi, my name is " + this.name + " and I'm a " +
this.occupation);
    }
};
person.introduce(); // "Hi, my name is John Doe and I'm a Developer"
```

4. Javascript Object Methods

- In this example, the **introduce** method use **this** keyword to reference the **name** and **occupation** properties of the object.
- Examples:

```
let shoppingCart = {
    items: [],
    addItem: function(item) {
        this.items.push(item);
    },
    removeItem: function(item) {
        let index = this.items.indexOf(item);
        if (index >= 0) {
            this.items.splice(index, 1);
        }
    },
```

4. Javascript Object Methods

```
    total: function() {
        let total = 0;
        for (let i = 0; i < this.items.length; i++) {
            total += this.items[i].price;
        }
        return total;
    }
};

shoppingCart.addItem({ name: "Shirt", price: 20 });
shoppingCart.addItem({ name: "Pants", price: 30 });
shoppingCart.addItem({ name: "Socks", price: 5 });
```

4. Javascript Object Methods

```
console.log(shoppingCart.items); // [{ name: "Shirt", price: 20 }, {
name: "Pants", price: 30 }, { name: "Socks", price: 5 }]
console.log(shoppingCart.total()); // 55

shoppingCart.removeItem({ name: "Pants", price: 30 });
console.log(shoppingCart.items); // [{ name: "Shirt", price: 20 }, {
name: "Socks", price: 5 }]
```

- In this example, the **shoppingCart** object has methods **addItem**, **removeItem** and **total**, that allows to add and remove items from the cart, or calculate the total amount of the items.

5. Javascript Object Inheritance

- In JavaScript, objects can inherit properties and methods from other objects, a mechanism known as object inheritance. This allows you to create new objects that are based on existing objects and can share their properties and methods.
- There are a few different ways to implement inheritance in JavaScript, but the most common one is using the prototype chain. Every object in JavaScript has a **prototype** property, which is a reference to the object from which it inherits properties and methods.
- When an object is created using an object constructor, the new object inherits properties and methods from the constructor's **prototype** property. You can also set the prototype of an object using the **Object.create()** method.

Javascript Object

5. Javascript Object Inheritance

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

5. Javascript Object Inheritance

- Here's an example of how you might use inheritance to create a Person object, and then create a **Student** object that inherits from it:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype.greet = function() {
    console.log("Hello, my name is " + this.name);
}

function Student(name, age, course) {
    Person.call(this, name, age);
    this.course = course;
}
```

5. Javascript Object Inheritance

```
Student.prototype = Object.create(Person.prototype);
Student.prototype.constructor = Student;

let student = new Student("John Smith", 25, "Computer Science");
student.greet(); // "Hello, my name is John Smith"
console.log(student.course); // "Computer Science"
```

- In this example, the **Person** object has a property **name**, a property **age**, and a method **greet**. The **Student** object inherits from **Person** by creating a new object that has **Person.prototype** as its prototype. With this, we can access the property **greet** from the **Student** object.
- Additionally, we set the **constructor** property of the **Student.prototype** to the Student function to make sure that **student instanceof Student** returns true.

5. Javascript Object Inheritance

- Here are a few more examples to illustrate different ways you can use inheritance in JavaScript:

2. Using class inheritance to create a Animal class and a Dog class that inherits from it:

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype.greet = function() {
    console.log("Hello, my name is " + this.name);
}
```

5. Javascript Object Inheritance

```
function Student(name, age, course) {
    Person.call(this, name, age);
    this.course = course;
}

class Animal {
    constructor(name) {
        this.name = name;
    }
    speak() {
        console.log(`${this.name} makes a noise.`);
    }
}
```

5. Javascript Object Inheritance

```
class Dog extends Animal {
    constructor(name, breed) {
        super(name);
        this.breed = breed;
    }
    speak() {
        console.log(`${this.name} barks.`);
    }
}

let dog = new Dog("Fido", "Golden Retriever");
dog.speak(); // Outputs, "Fido barks."
console.log(dog instanceof Dog); // true
console.log(dog instanceof Animal); // true
```

5. Javascript Object Inheritance

- In this example, we use class inheritance with **extends** keyword to create the **Dog** class that inherits from the **Animal** class. The **Dog** class has its own constructor and **speak()** method that overrides the one in the **Animal** class.
- In summary, Inheritance in javascript is the mechanism that allows objects to inherit properties and methods from other objects. the most common way to achieve this is by using the prototype chain. with this, we can create new objects that are based on existing objects and can share their properties and methods.

Javascript Object

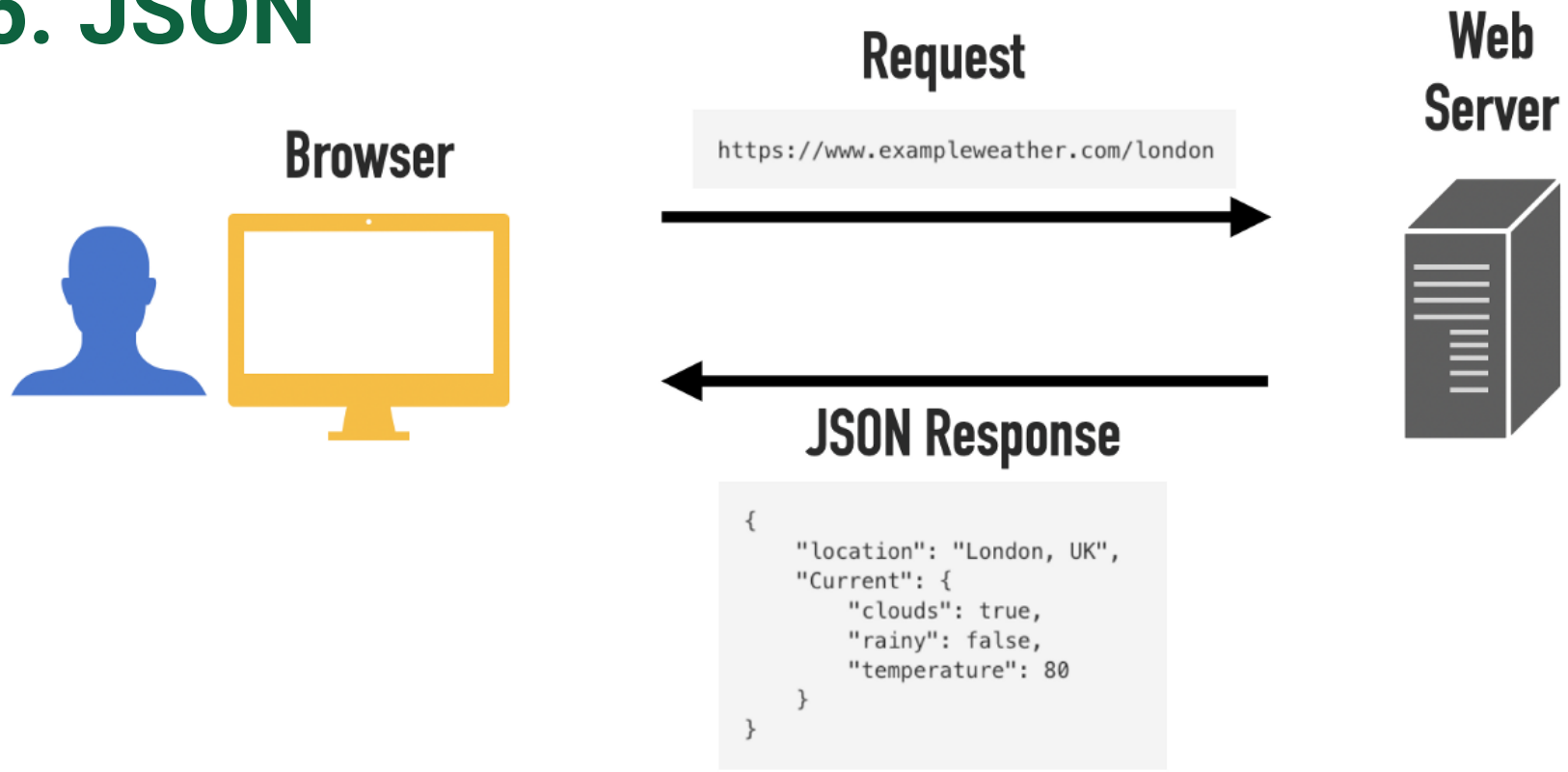
5. JSON

6. JSON

JSON Introduction

- JSON (JavaScript Object Notation) as a lightweight data-interchange format that is easy for humans to read and write, and easy for machines to parse and generate.
- JSON is language-independent and uses conventions that are familiar to programmers of the C family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others.
- JSON is widely used as **a way of communication between a server and a web application**, or between different parts of a web application.

6. JSON



- JSON is a subset of JavaScript, and its data structure is similar to JavaScript object literals, this means that JSON is based on a subset of the JavaScript Programming Language.

6. JSON

JSON Structure

A simple JSON object that represents a person:

```
{
 "name": "John Smith",
 "age": 25,
 "address": {
   "street": "Main St.",
   "city": "New York",
   "state": "NY"
 }
}
```

6. JSON

JSON Structure: the object

A simple JSON object that represents a person:

```
{ “key”: “value” }
```

JSON objects are made up of key-value pairs, where the keys must be strings and the values can be a string, number, boolean, null, array, or another JSON object.

- JSON objects are similar to JavaScript object literals, this means that the structure of JSON objects is similar to the structure of JavaScript objects, making it easy to use in JavaScript programs.
- JSON objects can contain nested JSON objects and arrays, this allows for complex data structures and hierarchical relationships between data.

6. JSON

Examples

This example illustrates the structure of a JSON object, with multiple key-value pairs, where the keys are strings and the values can be strings, numbers, or another JSON object.

1. A JSON object that represents a product with nested JSON objects:

```
{
 "productName": "Apple iPhone",
 "price": 799,
 "dimensions": {
   "height": 5.45,
   "width": 2.64,
   "depth": 0.30
 },
```

6. JSON

```
"reviews": [
   {
     "user": "Jane Doe",
     "rating": 4,
     "review": "Great phone but a bit expensive"
   },
   {
     "user": "John Smith",
     "rating": 5,
     "review": "Love it! Best phone I've ever had"
   }
 ]
}
```

- This example illustrates the use of nested JSON objects, with a JSON object within the main JSON object for the product details, such as price, dimensions, and reviews.

6. JSON

2. A JSON object that represents a collection of movie titles:

```
{
 "productName": "Apple iPhone",
 "price": 799,
 "dimensions": {
   "height": 5.45,
   "width": 2.64,
   "depth": 0.30
 },
```

6. JSON

```
{
 "movieTitles": [
   "The Shawshank Redemption",
   "The Godfather",
   "The Dark Knight",
   "The Lord of the Rings: The Return of the King",
   "Pulp Fiction"
 ]
}
```

- This example illustrates the use of arrays in JSON, where a JSON object has an array value that contains a collection of movie titles.

6. JSON

3. A JSON object that represents a collection of product details:

```
{
 "products": [
    {
      "name": "apple iphone",
      "price": 800
    },
    {
      "name": "samsung galaxy",
      "price": 600
    },
    {
      "name": "google pixel",
      "price": 700
    }
 ]
}
```

This example illustrates the use of arrays in JSON, where a JSON object has an array value that contains a collection of objects, each object has it's own properties name and price, showing the power of nested JSON.

6. JSONJSON on the Web

```
{
 "products": [
   {
     "name": "Apple iPhone",
     "price": 799
   },
{
     "name": "Samsung Galaxy",
     "price": 699
   },
   {
     "name": "Google Pixel",
     "price": 649
   }
 ]
}
```

JSON is often used to transmit data between a server and a web application, or between different parts of a web application.

Here is an example of a JSON file that stores a list of products and their prices, and how it could be used by a website to display the products and prices on the page:

This JSON file contains an array of objects, where each object represents a product, with properties such as "name" and "price". The file could be stored on the server and accessed by the website using JavaScript to retrieve the data and display it on the page.

6. JSON

Here's an example of how the JavaScript code might look like to retrieve the data from the JSON file and display it on the page:

```
//Retrieve the JSON file from the server
fetch('products.json')
  .then(response => response.json())
  .then(data => {
    //Access the products array
    let products = data.products;
    //Loop through the array and display each product
```

6. JSON

```
for(let i=0; i<products.length; i++) {
        let product = products[i];
        let productName = product.name;
        let productPrice = product.price;
        //Create a new element to display the product
        let productElement = document.createElement("div");
        productElement.innerHTML = productName + " - $" +
productPrice;
        //Add the element to the page
        document.getElementById("products").appendChild(product
Element);
    }
  });
```

6. JSON

- The previous slides code uses the fetch API to retrieve the JSON file from the server, and the .then method to parse the JSON data into a JavaScript object. Once the data is retrieved and parsed, it can be accessed using the dot notation, and the values of the product's properties can be accessed. The code loop through the array of products, creating a new element for each product and displaying its name and price, then adds the element to the page.
- It's worth noting that this is a simple example and in a real-world scenario, you will have to handle errors and also take into consideration the security aspects of the data.

6. JSON

JSON in Action

Code example of how to parse JSON data using Javascript

Here is an example of how to parse JSON data using JavaScript:

```
// JSON data
const jsonData = '{"name":"John Smith","age":30,"city":"New York"}';

// Parse JSON data
let data = JSON.parse(jsonData);

// Access the properties of the parsed JSON data
console.log(data.name); // "John Smith"
console.log(data.age); // 30
console.log(data.city); // "New York"
```

6. JSON

In this example, we have a JSON data as a string, which is assigned to the **jsonData** variable. Then, we use the **JSON.parse()** method to parse the JSON data into a JavaScript object. Once the data is parsed, we can access the properties of the object using the dot notation. As you can see, we access the properties **name**, **age** and **city** and log their values to the console.

6. JSON

You can also parse JSON data that is received from a server, by fetching it via an API call, for example:

```
fetch('https://api.example.com/data')
  .then(response => response.json())
  .then(data => {
    console.log(data); // JSON data
  });
```

- This code uses the **fetch** API to retrieve the JSON data from a server and the **.then** method to parse the JSON data into a JavaScript object. Once the data is retrieved and parsed, it can be accessed and used in the code.
- It's worth noting that in both examples, the JSON.parse() method will throw an error if the JSON data is not well-formed. It's always a good practice to add a try-catch block to handle errors.

Javascript Object

6. JSON

Code example of how to create a JSON Object in Javascript

```
// Creating a JavaScript object
let data = {
  name: "John Smith",
  age: 30,
  city: "New York"
};
// Convert the JavaScript object to JSON data
let jsonData = JSON.stringify(data);

console.log(jsonData); // '{"name":"John Smith","age":30,"city":"New
York"}'
```

In this example, we create a JavaScript object and assign it to the data variable. The object contains properties such as name, age, and city.

To convert the JavaScript object to JSON data, we use the **JSON.stringify()** method and pass the object as an argument. This method returns a JSON string, which can be saved to a file, sent to a server, or logged to the console.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

6. JSON

To convert the JavaScript object to JSON data, we use the **JSON.stringify()** method and pass the object as an argument. This method returns a JSON string, which can be saved to a file, sent to a server, or logged to the console.

6. JSON

You can also use the **JSON.stringify()** method to convert arrays and nested objects to JSON data:

```
let data = {
  name: "John Smith",
  age: 30,
  city: "New York",
  hobbies: ["reading", "traveling", "coding"],
  address: {
    street: "Main St",
    city: "New York",
    state: "NY"
  }
};
let jsonData = JSON.stringify(data);
console.log(jsonData);
```

This example shows how to convert an object that contains nested objects and arrays and convert it to json format.

It's worth noting that the JSON.stringify() method has an optional second argument that allows you to specify a replacer function that will be called for each item in the string, this can be useful for adding custom logic or filtering data before conversion.

Thank you for attending this presentation, I hope you have a better understanding of JavaScrip. If you have any questions, please feel free to reach out to me for further clarification.

FACULTY OF INFORMATION TECHNOLOGY

Universitas Advent Indonesia