Verse of the day

Proverbs 12:24:

"Diligent hands will rule, but laziness ends in forced labor."

FACULTY OF INFORMATION TECHNOLOGY

Universitas Advent Indonesia

Javascript Functions

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

Outlines

1: **Introduction**

- Title slide: "JavaScript Functions“
- Introduction to the topic of the presentation and its relevance

2: **What is a JavaScript Function?**

- Definition of a function in JavaScript
- Explanation of the syntax for declaring a function

3: **Function Parameters**

- Explanation of function parameters and how they are used
- Examples of declaring and calling functions with parameters

Outlines

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

Outlines

7: **Function Scope**

- Explanation of function scope and how it relates to variable scope
- Examples of how function scope affects variables and their accessibility

8: **Conclusion**

- Summary of the main points covered in the presentation
- Additional resources for further learning and reference

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

1. Introduction

JavaScript is a programming language that is widely used for creating interactive websites and web applications.

One of the fundamental building blocks of any JavaScript program is the function. In this presentation, we will explore what JavaScript functions are, how they work, and their many uses.

By the end of this presentation, you will have a solid understanding of functions in JavaScript and how to use them effectively in your own projects.

2. What is a JavaScript Function?

- A function in JavaScript is a block of code that can be executed when it is called.
- It can take input (known as arguments or parameters) and can return an output (known as a return value).
- Functions are reusable, meaning they can be called multiple times throughout a program.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

2. What is a JavaScript Function?

- The syntax for declaring a function in JavaScript is as follows:

```
function functionName(parameters) {
  // code to be executed
}
```

- **functionName** is the name of the function. It is used to call the function later on.
- **parameters** are the input values that can be passed to the function. They are optional and a function may not require any parameters at all.
- **{ }** encloses the code that will be executed when the function is called.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

2. What is a JavaScript Function?

An example of declaring and calling a function:

```
function greet(name) {
  console.log('Hello, ' + name);
}

greet('John'); // Output: 'Hello, John'
```

In this example, we declare a function named **greet** that takes one parameter **name** and logs a greeting to the console when called. Then we call that function by passing 'John' as an argument, and the output is 'Hello, John'

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

3. Function Parameters

- Function parameters are used to pass values into a function when it is called.
- These values can be used within the function to perform certain actions or calculations.
- When a function is called, the values passed in as arguments are matched with the corresponding parameter names in the function definition.

3. Function Parameters

```
function multiply(num1, num2) {
  return num1 * num2;
}

let result = multiply(5, 3);
console.log(result); // Output: 15
```

In this example, **num1** and **num2** are the parameters of the **multiply** function. When the function is called with the arguments 5 and 3, the value of **num1** is 5, and the value of **num2** is 3 within the function. This function take two parameters **num1** and **num2**, multiplies them and return the value, in this case it is 15 which is assigned to variable 'result'

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

3. Function Parameters

Function can also be defined without any parameters as well:

```
function printDate() {
  let today = new Date();
  console.log(today);
}
printDate();
```

In this example, **printDate** function is defined without any parameters. It creates a new date object and print the date to console when function is called.

4. Function Return Values

JavaScript functions can return a value using the return statement. The return statement ends the execution of the function and passes a value back to the calling code. For example,

```
function sum(a, b) {
  return a + b;
}
let result = sum(2, 3);
console.log(result); // Output: 5
```

In this example, the **sum** function takes two parameters, **a** and **b**, and returns their sum when called. The returned value is then assigned to the variable **result**, which is then logged to the console.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

4. Function Return Values

A function can also return more complex data types like objects or arrays:

```
function createPerson(name, age) {
  return {
    name: name,
    age: age
  };
}
let person = createPerson('John', 30);
console.log(person.name); // Output: 'John'
console.log(person.age); // Output: 30
```

In this example, the **createPerson** function takes two parameters, **name** and **age**, and returns an object containing those values. The returned object is then assigned to the variable **person**, which can be accessed by its properties **name** and **age**.

4. Function Return Values

It is also possible for a function to not return anything, in that case the returned value would be **undefined**.

```
function printName(name) {
  console.log(name);
}
let result = printName('John');
console.log(result); // Output: undefined
```

In this example, the **printName** function takes one parameter, **name**, and logs it to the console but it does not return any value, so the value of **result** is **undefined**.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

5. Anonymous Functions

- In JavaScript, functions can be declared without a name, these are known as anonymous functions.
- Anonymous functions are often used as arguments for higher-order functions (functions that take a function as an argument) or as callbacks (functions passed to another function to be executed at a later time).

5. Anonymous Functions

For example:

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(function(number) {
  return number * 2;
});
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

In this example, the **map** function is a higher-order function that takes a callback function as an argument. The callback function is an anonymous function that takes one parameter **number** and returns its double. This anonymous function is passed as an argument to the **map** function, which then applies the function to each element of the **numbers** array and returns a new array with the results.

5. Anonymous Functions

Another way to define Anonymous function is using arrow function expression

```
let button = document.querySelector('button');
button.addEventListener('click', function() {
  console.log('Button clicked');
});
```

In this example, an anonymous function is passed as a callback to the **addEventListener** method. The function will be executed when the button is clicked, logging the message to the console.

5. Anonymous Functions

It is also possible to assign anonymous function to a variable, and use that variable in place of function

```
let greet = function(){console.log("Hello")};
greet();
```

In this example, an anonymous function is assigned to variable **greet** and later used to call function.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

6. Arrow Functions

- Arrow functions, also known as "fat arrow" functions, are a shorthand notation for declaring anonymous functions in JavaScript, introduced in ECMAScript 6.
- Arrow functions have a shorter syntax than traditional function declarations, making them more convenient to use in certain situations.

6. Arrow Functions

Here is the comparison between traditional function declaration and arrow function

```
// Traditional function declaration
function sum(a, b) {
  return a + b;
}

// Arrow function
let sum = (a, b) => {
  return a + b;
}
```

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

6. Arrow Functions

When the function has only one expression and does not use **this**, **super**, **arguments**, or **new.target**, the curly braces and the **return** keyword can be omitted:

```
let sum = (a, b) => a + b;
```

Arrow functions can also be used as arguments for higher-order functions and as callbacks, just like anonymous functions.

```
let numbers = [1, 2, 3, 4, 5];
let doubledNumbers = numbers.map(number => number * 2);
console.log(doubledNumbers); // Output: [2, 4, 6, 8, 10]
```

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

6. Arrow Functions

One important thing to note is that arrow functions do not bind their own **this** value, they inherit **this** from the surrounding scope. This can be useful in situations where you want to access a parent context's **this** inside a callback function, but can also lead to unexpected behavior if **this** is used in the callback and the parent context's **this** value is not what you expect.

```
let obj = {
  name: "obj",
  sayName: () => {console.log(this.name)}
}
console.log(obj.sayName()) //undefined
```

6. Arrow Functions

```
let obj = {
  name: "obj",
  sayName: () => {console.log(this.name)}
}
console.log(obj.sayName()) //undefined
```

- In this example, **this** inside the function refer to the global scope instead of the obj scope, so the output is **undefined**.
- It's worth noting that arrow function have some limitation with respect to 'this' binding and 'arguments' object.

7. Function Scope

- In JavaScript, variable scope refers to the accessibility of variables in different parts of a program.
- Functions create their own scope, which means that variables declared inside a function are not accessible outside of that function, and variables declared outside of a function are not accessible inside the function, unless they are passed as arguments or are declared as global variables.

7. Function Scope

Function scope is created when a function is defined and destroyed when the function is executed. Variables declared within the function are only accessible within the function and not outside of it.

```
function sayHello() {
  let message = "Hello";
  console.log(message);
}
sayHello(); // Output: "Hello"
console.log(message); // ReferenceError: message is not defined
```

In this example, the variable **message** is declared within the **sayHello** function and is only accessible within that function. Attempting to access the variable outside of the function causes a **ReferenceError**.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

7. Function Scope

Function scope also applies to variables with the same name, declared in different scopes:

```
let name = "John";
function sayName() {
  let name = "Jane";
  console.log(name);
}
sayName(); // Output: "Jane"
console.log(name); // Output: "John"
```

In this example, there are two variables with the same name, one declared in the global scope and one declared within the **sayName** function. The variable declared within the function takes precedence and the global variable is shadowed.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

7. Function Scope

Function scope can also affect the behavior of variables that are passed as arguments:

```
let x = 10;
function add(y) {
  y = y + 5;
  return y;
}
let result = add(x);
console.log(x); // Output: 10
console.log(result); // Output: 15
```

In this example, the variable **x** is passed as an argument to the **add** function. Within the function, the value of **x** is stored in a new variable **y** which is created in the function's scope, so the change made to **y** inside the function does not affect the value of **x** outside of the function.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

6. Function Scope

Function scope can also be nested, where a function can be defined inside another function and the inner function will have access to the variables of the outer function.

```
function outer(){
  let x = 10;
  function inner(){
    console.log(x);
  }
  inner();
}
outer(); // Output: 10
```

In this example, **inner** function has access to variable **x** declared inside the **outer** function, as it is in its scope and also can be accessed by outer function as well.

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

8. Conclusion

In this presentation, we covered the basics of JavaScript functions, including their syntax, parameters, return values, scope, and usage with anonymous and arrow functions.

Functions are a fundamental building block of JavaScript, allowing for the creation of reusable code and the separation of concerns within a program. Understanding the concept of function scope and how it relates to variable scope is important for writing maintainable and efficient code.

8. Conclusion

We have discussed various ways of declaring function in JavaScript, from traditional function declaration, anonymous function, and arrow function. We also discussed how they can be used as callbacks and with higher-order functions.

To continue learning about JavaScript functions, I would recommend checking out the following resources:

- The Mozilla Developer Network's JavaScript Guide: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
- Eloquent JavaScript, a popular book on JavaScript programming: https://eloquentjavascript.net/03_functions.html
- JavaScript.info, a comprehensive resource for learning JavaScript: https://javascript.info/function

FACULTY OF INFORMATION TECHNOLOGY
Universitas Advent Indonesia

Thank you for attending this presentation, I hope you have a better understanding of JavaScript functions and their uses.

If you have any questions, please feel free to reach out to me for further clarification.