Functions

Muhammad Zahir Junejo

Lecture - Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
  - Please review Code of Conduct (in Student Undertaking Agreement) if unsure
- No question is daft or silly - **ask them!**
- Q&A session at the end of the lesson, should you wish to ask any follow-up questions.
- Should you have any questions after the lecture, please schedule a mentor session.
- For all non-academic questions, please submit a query: www.hyperiondev.com/support

Lecture Objectives

1. Define functions and understand their role in code organization.
2. Call functions with arguments and handle returned values.
3. Grasp the concepts of scope and how it affects variable visibility.
4. Explore arrow functions and their benefits for concise coding.

What are Functions?

- A function is a reusable block of code that performs a specific task. Functions allow us to encapsulate logic, improve code organization, and enable code reusability.
- Syntax:

```
function functionName(parameters) {
  // Function body
  // Perform tasks here
}
```

Defining Functions

- Declare a function using the function keyword.
- Syntax:

```
function functionName(parameters) {
  // Function body
  // Perform tasks here
}
```

- Assign a function to a variable, function expressions.

```
const add = function(x, y) {
  return x + y;
};
```

Calling Functions

- Call a function by using its name followed by parentheses.

```
greet();
```

- Functions can take parameters, which are values passed to them when called.

```
const sum = add(5, 3);
console.log(sum); // Output: 8
```

Return statements

- Functions can return values using the return statement. The function ends when a return statement is encountered.

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

- You can use the returned value of a function in expressions or assignments.

```
const result = multiply(4, 6);
console.log(result); // Output: 24
```

Scope

- Scope refers to the context in which variables and functions are defined and accessed.
- Local function scope: Variables defined within a function are only accessible within that function's scope.

```
function example() {
  const localVar = 'I am local';
  console.log(localVar);
}
```

Global Scope

- Variables declared outside of any function are considered to be in the global scope.

```
const globalVar = 'I am global';
function example() {
  console.log(globalVar);
}

example(); // Output: I am global
```

- Limit global variables to minimize potential issues and maintain cleaner code.

Function Scope Interaction

- Inner scopes have access to variables defined in outer scopes, but not vice versa.

```
function outer() {
  const outerVar = 'I am from outer';

  function inner() {
    console.log(outerVar);
  }

  inner(); // Output: I am from outer
}
```

Function Hoisting

- In JavaScript, function declarations are hoisted to the top of their scope, allowing you to call them before they're defined.

```
greet('Alice'); // Output: Hello, Alice!

function greet(name) {
  console.log(`Hello, ${name}!`);
}
```

Arrow Functions

- Arrow functions are a concise way to define functions in JavaScript.
- Arrow functions have a shorter syntax using the => arrow notation.

```
const add = (x, y) => x + y;
```

- If the function takes only one parameter, you can omit the parentheses.

```
const square = x => x * x;
```

- If the function doesn't take any parameters, you still need to include empty parentheses.

```
const greet = () => 'Hello, world!';
```

References

- https://www.programiz.com/javascript/ES6
- https://www.programiz.com/javascript/function

Questions and Answers

Thank You!