Lesson 5: Javascript Control Flow and Functions

Control Flow

Control flow in JavaScript refers to the order in which statements and expressions are executed in a program. It determines the path that the program takes based on certain conditions and loops. Control flow allows you to make decisions, repeat actions, and control the flow of execution.

In JavaScript, control flow is managed using control structures such as conditionals (if...else statements, switch statements) and loops (for loops, while loops, do...while loops). These control structures enable you to alter the flow of execution based on specific conditions or repeatedly execute blocks of code.

Conditionals:

Conditionals allow you to execute different blocks of code based on specific conditions.

1. The most commonly used conditional statement is the **if...else** statement. It evaluates a condition and executes a block of code if the condition is true, and an alternative block of code if the condition is false. Here's an example:

```
let hour = 15;
let greeting;

if (hour < 12) {
  greeting = "Good morning!";
} else if (hour < 18) {
  greeting = "Good afternoon!";
} else {
  greeting = "Good evening!";
}

console.log(greeting);
```

2. **switch Statement:** The switch statement is used to select one of many code blocks to be executed based on the value of an expression. It provides a concise way to handle multiple cases. Here's an example:

```
let day = 3;
let dayName;

switch (day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  // ...
  default:
    dayName = "Unknown";
    break;
}

console.log(dayName);
```

3. **for Loop:** The for loop allows you to repeatedly execute a block of code for a specified number of times. It consists of an initialization, a condition, and an iteration statement. Here's an example:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

4. **while Loop:** The while loop executes a block of code as long as a specified condition is true. It checks the condition before each iteration. Here's an example:

```
let i = 0;

while (i < 5) {
  console.log(i);
  i++;
}
```

5. **do...while Loop:** The do...while loop is similar to the while loop but guarantees that the code block is executed at least once before checking the condition. Here's an example:

```
let i = 0;

do {
  console.log(i);
  i++;
} while (i < 5);
```

Control flow structures allow you to control the flow of your program and make it more dynamic and adaptable.

Functions

Functions in JavaScript are reusable blocks of code that perform specific tasks. They allow you to organize and modularize your code, making it more maintainable and reusable. Functions can take input, called parameters or arguments, and can return a value as output.

1. **Defining Functions:**
   In JavaScript, you can define a function using the function keyword, followed by the function name, a pair of parentheses for parameters (if any), and a block of code enclosed in curly braces. Here's an example:

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

In this example, **greet** is the name of the function, and **name** is the parameter. The code block inside the function will be executed when the function is called.

2. **Calling Functions:**
   To execute a function and run its code, you need to call it. You can call a function by using its name followed by parentheses and passing arguments (if any) within the parentheses. Here's an example of calling the greet function:

```
greet("John");
```

In this case, the string "John" is passed as an argument to the **greet** function. The function will be executed, and it will log the message "Hello, John!" to the console.

3. **Function Parameters and Arguments:**
   Functions can accept parameters, which act as placeholders for values that will be passed to the function when it is called. Parameters are declared within the parentheses of the function declaration. Here's an example:

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

In this example, the **multiply** function has two parameters, **a** and **b**. When the function is called, you need to provide arguments that match the number and order of the parameters. For example:

```
let result = multiply(3, 4);
console.log(result); // Output: 12
```

Here, the values **3** and **4** are passed as arguments to the **multiply** function. The function multiplies these values and returns the result, which is then stored in the **result** variable and logged to the console.

4. **Return Statement:**

Functions can also return values using the return statement. The return statement specifies the value that the function should return when it is called. Here's an example:

```
function square(number) {
  return number * number;
}

let result = square(5);
console.log(result); // Output: 25
```

In this example, the **square** function takes a **number** parameter, squares it, and returns the result. When the function is called with the argument **5**, it returns **25**, which is then stored in the **result** variable and logged to the console.

5. **Function Expressions and Arrow Functions:**
   In addition to the traditional function declaration, JavaScript also supports function expressions and arrow functions.

- Function Expressions:

A function expression assigns a function to a variable. Here's an example:

```
const greet = function(name) {
  console.log("Hello, " + name + "!");
};

greet("John");
```

In this example, the function expression is assigned to the variable **greet**. The function can then be called using the variable name, followed by parentheses.

- Arrow Functions:

Arrow functions provide a more concise syntax for defining functions. They use the => syntax and have implicit return. Here's an example:

```
const multiply = (a, b) => a * b;
```

```
let result = multiply(3, 4);
console.log(result); // Output: 12
```

In this example, the arrow function **multiply** takes two parameters, **a** and **b**, and returns their product. The arrow function is assigned to the variable **multiply**, and it can be called using the variable name, followed by parentheses.

Functions play a crucial role in JavaScript programming. They enable code reuse, modularization, and help in organizing complex logic into manageable units. By utilizing functions, you can write cleaner, more maintainable, and reusable code.