JavaScript

Statements

What are Statements

Statements are the basic building blocks of JavaScript programs

Statements in JavaScript are similar to sentences in English language

A statement must expresses a complete thought or action

A statement must perform a specific action or task

A statement is executed in a specific order

A statement must terminate with a semicolon (;)

Writing Javascript Statements

Statements are written on separate lines and must be terminated with a semicolon.

JavaScript Statements are case-sensitive.

Statements can span multiple lines and can contain expressions and comments.

Blocks can be used to group statements together.

Conditional statements only execute if a certain condition is true.

Iterative statements execute repeatedly until a certain condition is met.

Function declarations have a specific keyword to indicate their purpose.

Statements can manipulate and interact with data stored in variables and objects.

Statements can interact with the DOM to modify web page content.

JavaScript has strict mode to enforce stricter rules for writing statements.

Statements can include various types of operators to perform actions on data.

Statements can include control flow constructs such as loops and conditionals to control program flow.

Categories of Statements

1. Declaration statements:used to declare and initialize variables, functions, classes, and constants
2. Expression statements: used to evaluate and execute expressions, these are expressions
3. Control flow statements: that control the flow of the program's execution based on specific conditions
4. Exception handling statements: used to handle errors and unexpected events that may occur during the execution of a program
5. Debugging statements: Console statements and Debugger keyword are for identifying and fixing errors in code
6. Directive statements: used to provide special instructions to the JavaScript engine eg. “use strict”

We would be focusing more on the **control flow statements** in the course of this lecture.

Control flow statements

1. Conditional statements: These statements allow developers to execute different blocks of code depending on whether a certain condition is true or false. the if statement and the switch statement.

2. Loop statements: These statements allow developers to execute the same block of code repeatedly until a certain condition is met. for loop, while loop,do-while loop

3. Jump statements: These statements allow developers to break out of loops or skip certain iterations based on certain conditions. Examples include "break" statements and "continue" statements.

4. Error handling statements: These statements allow developers to handle exceptions or errors that may occur during the execution of their code. Examples include Try...catch statements, throw statements

Conditional Statements

If, else, else if, switch

The if statement

The if statement is used to execute a block of code if a specified condition is true. The basic syntax:

```
if (condition) {
  // code to be executed if condition is true
}
```

The condition is a boolean expression that evaluates to either true or false.

If the condition is true, the code inside the curly braces is executed. If the condition is false, the code is skipped over and execution continues with the next statement.

The Else Statement

You can also include an else clause to execute code if the condition is false:

```
if (condition) {
// code to execute if condition is true
} else {
// code to execute if condition is false
}
```

Example:

```
let num = 10;

if (num > 20) {
  console.log("Num is greater than 20");
} else {
  console.log("Num is less than or equal to 20");
}
```

The Else if Statements

Additionally, you can chain multiple if-else statements together to handle more complex logic:

```
if (condition1) {
// code to execute if condition1 is true
} else if (condition2) {
// code to execute if condition2 is true
} else {
// code to execute if all conditions are false
}
```

Remember to always include parentheses around the condition, and to use curly braces to group the code to be executed inside the if statement

Example:

```
let grade = 70;

if (grade >= 90) {
  console.log("You got an A!");
} else if (grade >= 80) {
  console.log("You got a B!");
} else if (grade >= 70) {
  console.log("You got a C!");
} else {
  console.log("You failed the class.");
}
```

The Switch Statement

Switch provides a concise way to execute multiple conditions based on a single expression. This is the syntax:

```
switch(expression) {
  case value1:
    // code block
    break;
  case value2:
    // code block
    break;
  case value3:
    // code block
    break;
  default:
    // code block
}
```

```
let dayOfWeek = 2;
let dayName;

switch(dayOfWeek) {
  case 1:
    dayName = 'Monday';
    break;
  case 2:
    dayName = 'Tuesday';
    break;
  case 3:
    dayName = 'Wednesday';
    break;
  case 4:
    dayName = 'Thursday';
    break;
  case 5:
    dayName = 'Friday';
    break;
  case 6:
    dayName = 'Saturday';
    break;
  case 7:
    dayName = 'Sunday';
    break;
  default:
    dayName = 'Invalid day';
}

console.log(dayName); // Output: Tuesday
```

Loop Statements
for, while, do while

Different flavours of the for loop

1. The basic for loop: allows you to execute a block of code repeatedly
2. The for...in loop: This loop iterates over the properties of an iterable object (keys)
3. The for...of loop: This loop iterates over the (values) of an iterable object
4. The forEach() method: This is an array method is used on arrays to execute a provided function once for each element in the array

The Basic for loop

```
for (initialization; condition; increment/decrement) {
  // code to be executed
}
```

Example:

```
const sales = [120, 150, 180, 200, 175, 140, 160];
let totalSales = 0;

for (let i = 0; i < sales.length; i++) {
  totalSales += sales[i];
}
console.log(`Total sales for the week: ${totalSales}`);
```

The initialization expression is executed once before the loop starts
The condition expression is evaluated at the beginning of each loop iteration and determines whether the loop should continue or not
The increment/decrement expression is executed at the end of each loop iteration and is used to update the loop variable.

The for in loop

```
for (variable in object) {
  // code to be executed
}
```

Example:

```
const person = {
  name: 'John',
  age: 30,
  city: 'New York'
};

for (let prop in person) {
  console.log(prop + ': ' + person[prop]);
}
```

The for of loop

```
for (variable of iterable) {
  // code to be executed
}
```

Example:

```
const numbers = [1, 2, 3, 4, 5];
for (let i = 0; i < numbers.length; i++) {
  console.log(numbers[i]);
}
```

The for each loop

Let Array = [a,b,c,d,e];

array.forEach(function(currentValue, index, arr), thisValue)

The second argument “thisValue” is optional and refers to the value of “this”. We would talk more about it once we get to objects.

Example:

```
let numbers = [1, 2, 3, 4, 5];
numbers.forEach(function(number) {
  console.log(number);
});
```

Note that the forEach() loop cannot be used to iterate over objects. It is specifically designed for arrays

The while loop

```
while (condition) {
  // code to be executed
}
```

Example:

```
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
```

The do while loop

```
do {
  // code block to be executed
} while (condition);
```

Example:

```
let number;
do {
  number = prompt("Please enter a number:");
} while (isNaN(number));

console.log("The number you entered is: " + number);
```

The do-while loop is similar to the while loop, but with a slight difference. In a do-while loop, the block of code is executed first, and then the condition is checked. If the condition is true, the block of code is executed again.

Jump Statements

break, continue and return

break

The break statement is used to terminate a loop or switch statement.

```
while (condition) {
  // code to execute
  if (some condition) {
    break;
  }
}
```

Example:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break;
  }
  console.log(i);
}
```

continue

the continue statement is used within loops to skip the current iteration of the loop and move on to the next one

```
while (condition) {
  if (some condition) {
    continue;
  }
}
```

Example:

```
for (let i = 0; i < 10; i++) {
  if (i === 3) {
    continue; // skips iteration 3 and jumps to 4
  }
  console.log(i);
}
```

return

The return statement is used in JavaScript functions to return a value and terminate the function's execution.

```
while (condition) {
  if (some condition) {
    continue;
  }
}
```

Example:

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

console.log(sum(2, 3)); // Output: 5
```

Error Handling Statements

try…catch and throw

Try…catch Statement

The try...catch statement is used to try a block of code and catch any errors that occur.

```
try {
  // block of code to try
} catch (error) {
  // code to handle the error
}
```

The catch block contains the code to handle the error, which typically includes logging the error or displaying an error message to the user.

Throw Statement

The throw statement is used to explicitly throw an error from within a block of code

Syntax: throw new Error('Error message goes here');

Example:

```
function divide(num1, num2) {
  if (num2 === 0) {
    throw new Error("Cannot divide by zero");
  }
  return num1 / num2;
}

try {
  let result = divide(10, 0);
  console.log("Result:", result);
} catch (error) {
  console.error("Error:", error.message);
}

divide();
```

Debugging

Understanding error messages

1. Error: A mistake or problem in the code that prevents it from executing correctly.
2. Exception: An object that represents an error or exceptional condition that occurs during the execution of the code.
3. Bug: A problem in the code that causes unexpected behavior or incorrect results.
4. Debugging: The process of identifying, analyzing, and fixing errors or bugs in the code.
5. Syntax Error: An error that occurs when the code violates the grammar or syntax rules of the programming language.
6. Runtime Error: An error that occurs during the execution of the code, typically caused by unexpected input or incorrect logic.
7. Logic Error: An error that occurs when the code is syntactically correct and runs without throwing errors, but produces incorrect results.
8. Stack Trace: A report that shows the call stack of a program at a specific point in time, often used to debug errors.
9. Breakpoint: A point in the code where the debugger will stop execution to allow for inspection of variables and other program state.
10. Assertion: A statement that checks a condition and raises an exception if it is false, often used to catch logic errors during debugging.

Methods of Debugging in Javascript

1. Console logging: Using console.log() statements to log messages to the browser console, which can help you trace the flow of your code and identify errors.
2. Debugger statements: Using debugger statements in your code to pause execution and allow you to step through your code line by line.
3. Browser DevTools: Using browser developer tools, such as the Chrome DevTools or Firefox Developer Edition, to debug your code. These tools allow you to inspect HTML and CSS, set breakpoints, and step through code.
4. Code editor plugins: Some code editors, such as Visual Studio Code, have built-in debugging tools or plugins that allow you to debug your code within the editor.
5. Third-party debugging tools: There are also third-party debugging tools, such as the JavaScript debugger tool from the Mozilla Developer Network, that can help you debug your code.
6. Unit tests: Writing unit tests can help you catch errors before they occur and ensure that your code is working as expected.

NB: It's important to use a combination of these methods to effectively debug your code and catch any errors or issues.

End

Javascript statements

Group Work

1. Guessing game: Create a JavaScript program that generates a random number between 1 and 10 and prompts the user to guess the number. The program should give hints such as "higher" or "lower" until the user correctly guesses the number.
2. FizzBuzz: Write a JavaScript program that prints the numbers from 1 to 100, but for multiples of three, print "Fizz" instead of the number, and for multiples of five, print "Buzz". For numbers that are multiples of both three and five, print "FizzBuzz".
3. Write a program that uses a for loop to iterate through an array of numbers and calculates the average of the numbers. Print it out to the console.

4. Write a program that prompts the user to enter a phone number and checks if the phone number is an MTN number or Airtel Number or other networks using if statements.

5. Write a program that prompts the user to enter a string and then checks if it is a palindrome. A palindrome is a word or phrase that reads the same backward as forward. Example (level, madam, radar, refer, civic);

Over the weekend, write

1. A program that determines whether a given number is odd or even.
2. A program that calculates the sum of the first n numbers in a sequence.
3. A program that checks if a given year is a leap year.
4. A program that converts a given temperature from Fahrenheit to Celsius or vice versa.
5. A program that determines if a given string is a palindrome.
6. A program that sorts an array of numbers in ascending or descending order.
7. A program that calculates the factorial of a given number.
8. A program that checks if a given number is prime or composite.
9. A program that converts a given decimal number to binary or vice versa.
10. A program that finds the largest and smallest elements in an array.
11. A program that calculates the average of a set of numbers.
12. A program that determines if a given string is a pangram.
13. A program that generates the Fibonacci sequence up to a given number of terms.
14. A program that determines if a given string is a valid email address.
15. A program that checks if a given number is a perfect square.
16. A program that converts a given string to title case.
17. A program that counts the number of words in a given string.
18. A program that determines if a given year is a leap year, and if not, outputs the next leap year.