CERTIFIED QUIZ

Introduction to JavaScript

JavaScript is a high-level, interpreted programming language that is widely used for creating dynamic and interactive web applications. Originally developed by Brendan Eich and first released in 1995, JavaScript has evolved significantly and is now a core technology of the World Wide Web, alongside HTML and CSS. It enables developers to implement complex features on web pages, such as real-time content updates, interactive forms, animations, and multimedia integration. JavaScript is supported by all modern web browsers and can be run on the server-side using environments like Node.js. JavaScript follows a multi-paradigm approach, supporting event-driven, functional, and imperative programming styles. This flexibility makes it one of the most popular and versatile programming languages in use today.

Basic Syntax

JavaScript syntax refers to the set of rules that define a correctly structured JavaScript program. Key components include:

- Statements: Instructions executed by the browser. Each statement typically ends with a semicolon (;), though JavaScript does not require them in all cases.
- Comments: Used to describe code and can be single-line (//) or multi-line (/* /).
- *Variables: Declared using the var, let, or const keywords. var is function-scoped, while let and const are block-scoped.

Example:
javascript
let message = 'Hello, World!';
console.log(message); // Output: Hello, World!

Data Types and Variables

JavaScript supports several data types that can be divided into two categories: primitive and reference types. Primitive types include:

- Number: Represents both integers and floating-point numbers.
- String: Represents text data.
- Boolean: Represents logical entities: true or false.
- Null: Represents an intentional absence of value.
- Undefined: Indicates a variable that has been declared but has not yet been assigned a value.
- Symbol: A unique, immutable data type introduced in ES6 (ECMAScript 2015).

Reference types include objects, arrays, and functions.

Variables can be declared using var, let, or const. Use const when you don't plan to reassign the variable, and use let for variables that will change. Avoid var as it has function scope and can lead to unpredictable behavior.

Operators

JavaScript includes a variety of operators to perform different operations on variables and values:

- Arithmetic Operators: +, -, , /, % (modulus)
- *Assignment Operators: =, +=, -=, =, /=
- *Comparison Operators: ==, ===, !=, !==, >, <, >=, <=
- Logical Operators: && (AND), || (OR), ! (NOT)

- Ternary Operator: condition ? expression1 : expression2

Example:
javascript
```
let x = 10;
let y = 20;
console.log(x + y); // Output: 30
console.log(x > y); // Output: false
```

Functions

Functions in JavaScript are blocks of code designed to perform a specific task. Functions are executed when they are called or invoked. A function can take parameters and return a value.

- Function Declaration:
javascript
```
function greet(name) {
return 'Hello, ' + name;
}
console.log(greet('Alice')); // Output: Hello, Alice
```

- Function Expressions:
javascript
```
const greet = function(name) {
return 'Hello, ' + name;
};
```

- Arrow Functions (introduced in ES6):
javascript
```
const greet = (name) => 'Hello, ' + name;
```

Arrow functions have a concise syntax and do not bind their own this.

Control Structures

JavaScript provides several control structures for managing the flow of the program:

- Conditional Statements:
- if, else if, else: Used for decision-making.
javascript
```
let num = 10;
if (num > 5) {
console.log('Greater than 5');
} else {
console.log('5 or less');
}
```

- switch: Used to perform different actions based on different conditions.
javascript
```
let fruit = 'apple';
switch (fruit) {
case 'apple':
console.log('Apple');
break;
case 'banana':
console.log('Banana');
break;
default:
console.log('Unknown fruit');
}
```

- Loops:
- for, while, do...while: Used to repeat a block of code.

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

Objects and Arrays

Objects in JavaScript are collections of key-value pairs, while arrays are ordered lists of values:

- Objects:

javascript
```
const person = {
name: 'John',
age: 30,
greet: function() {
console.log('Hello');
}
};
console.log(person.name); // Output: John
person.greet(); // Output: Hello
```

- Arrays:

javascript
```
const numbers = [1, 2, 3, 4];
console.log(numbers[0]); // Output: 1
```

Arrays are zero-indexed, and methods such as push(), pop(), shift(), unshift(), map(), and filter() can be used to manipulate them.

DOM Manipulation

JavaScript can interact with the Document Object Model (DOM) to dynamically change content, structure, and styles on a web page. Common methods include:

- getElementById: Returns an element by its ID.

javascript
```
document.getElementById('myElement').innerHTML = 'New Content';
```

- querySelector: Returns the first element that matches a specified CSS selector.

javascript
```
document.querySelector('.myClass').style.color = 'blue';
```

- createElement: Creates a new HTML element.

javascript
```
const newElement = document.createElement('div');
```

DOM manipulation is widely used to build interactive web applications.

Events

JavaScript handles user interactions through events. Event listeners can be attached to elements to detect and respond to user actions like clicks, key presses, or form submissions:

- Event Listener:

```
javascript
document.getElementById('myButton').addEventListener('click', function() {
alert('Button clicked!');
});
```

Events like click, mouseover, keydown, and submit are commonly used in web development.

Asynchronous JavaScript

JavaScript is single-threaded but can handle asynchronous operations like making API requests, timers, or handling events without blocking the main execution thread:

- Callbacks: Functions passed as arguments to be executed later.

```
javascript
setTimeout(function() {
console.log('Hello after 2 seconds');
}, 2000);
```

- Promises: Objects representing the eventual completion or failure of an asynchronous operation.

```
javascript
const promise = new Promise((resolve, reject) => {
setTimeout(() => resolve('Done!'), 3000);
});
promise.then((message) => console.log(message));
```

- async/await: Introduced in ES8 for handling promises in a more readable way.

```
javascript
async function fetchData() {
const response = await fetch('https://api.example.com/data');
const data = await response.json();
console.log(data);
}
```

Error Handling

JavaScript handles errors through try...catch blocks:

```
javascript
try {
// Code that may throw an error
let result = riskyOperation();
} catch (error) {
console.error('Error:', error.message);
} finally {
console.log('This runs regardless of error.');
}
```

This ensures that the program can gracefully handle exceptions without crashing.

JavaScript Versions (ES6 and Beyond)

JavaScript has undergone significant updates, particularly with ECMAScript 6 (ES6/ES2015) and beyond. Key features introduced include:

- let and const for variable declarations.
- Arrow functions for a more concise syntax.
- Template literals for embedding expressions in strings using backticks.
- Destructuring for easier extraction of values from arrays or objects.
- Modules for splitting code into reusable pieces.

Example of destructuring:
javascript
```
const [a, b] = [1, 2];
console.log(a); // Output: 1
```