Assignment on control structures by Zion

Control structures are essential components of programming languages that direct the execution flow of a program. In JavaScript, control structures enable developers to make decisions, iterate through data, handle errors, and create complex algorithms.

Here's a breakdown of the key control structures in JavaScript:

1. Conditional Statements:

· **if...else:** This is the most fundamental control structure. It checks a condition and executes a block of code if the condition is true. Optionally, you can include an `else` block to execute code if the condition is false.

Example

```
let age = 18;



if (age >= 18) {

  console.log("You are eligible to vote.");

} else {

  console.log("You are not eligible to vote.");

}
```

· **switch:** This structure is used for multi-way branching based on a single expression's value. It compares the expression with different cases and executes the associated code block if there's a match. A default case can be included for unmatched values.

Example

```
let day = "monday";



switch (day) {

  case "monday":

       console.log("Start of the week!");

       break;

  case "weekend":

       console.log("Time to relax!");

       break;

  default:

       console.log("It's a weekday.");

}
```

2. Loops:

· **for:** This loop is ideal for iterating a predetermined number of times. It has three parts: initialization, condition, and increment/decrement. The code block runs as long as the condition remains true, and the expression updates the counter after each iteration.

Example

```
for (let i = 0; i < 5; i++) {

  console.log("Iteration", i + 1);

}
```

· **while:** This loop keeps executing a code block as long as a specified condition evaluates to true. It's useful when the number of iterations isn't known beforehand.

Example

```
let count = 0;

while (count < 3) {

  console.log("Count:", count);

  count++;

}
```

· **do...while:** Similar to `while`, this loop guarantees at least one execution of the code block before checking the condition.

Example

```
let count = 0;

do {

  console.log("Count:", count);

  count++;

} while (count < 0); // This condition will never be true, but the loop runs once
```

· **Nested Loops**

Nested loops involve one
loop inside another, enabling developers to handle complex iterations and manipulate multidimensional data structures.

3. Other Control Structures:

· **Ternary Operator:** This is a shorthand way of writing an if-else statement in a single line. It evaluates a condition and returns one of two expressions based on the outcome.

Example

```
let message = age >= 18 ? "You can vote" : "You cannot vote";
```

```
console.log(message);
```

- **break and continue:** These statements are used within loops to control the flow. `break` exits the loop completely, while `continue` skips the current iteration and moves to the next.

By mastering these control structures, you'll gain the power to create complex logic and interactivity in your JavaScript programs. For a deeper understanding, consider exploring online tutorials and practicing writing code that incorporates different control structures.