Flow Control Statement

Flow control statements are used to change the flow of execution of the program.

1. if
2. if else
3. else if / ladder if
4. nested if

1. if

`if` will check a condition and when condition is true then `if` block will execute otherwise if block will not execute.

Syntax:

```
if (condition) {
    //if block
}
// outside statement
```

2. if else

`if else` is used to check a condition and when condition is true then `if` block is execute otherwise `else` block will execute.

Syntax:

```
if (condition) {
    // if block
} else {
    //else block
}
```

3. else if/ladder if

It is used to check multiple conditions and when a condition found true then `if` block will execute. If no condition is true then `else` block will execute.

Syntax:

```
if (condition 1) {
    //if block-1
} else if (condition 2) {
    // if block-2
} else if (condition 3) {
    //if block -3
}
.
.
.
else {
    //else block
}
```

4. Nested if

When one `if` block is used inside another inside block then it is called `Nested if` block.

Syntax:

```
if (condition - 1) {
    if (condition - 1.1) {
        //if block
    } else {
        //else block
    }
} else {
    if (condition - 1.1) {
        //if block
    } else {
        //else block
    }
}
```

Loop

Loop is block of instruction which executes until loop condition will be true. It is of two types:

- Entry control loop
- Exit control loop

Loop is faster than recursion.

Entry control loop

If the loop condition is at entry point and after loop condition there is a loop body then it is called entry control loop.

Ex: `while` loop and `for` loop

Exit control loop

If the loop condition is at exit but before loop condition there is a loop body then it is called exit control loop.

Ex: `do while` loop

1. while loop

`while` loop is a entry control loop which will execute repeatedly until loop condition will be true.

Syntax:

```
while (condition) {
    // loop body

    // increment or decrement body
}
```

Example:

```
let i = 1;

while (i <= 5) {
    console.log("i : ", i);
    i++;
}
// output
1
2
3
4
5
```

2. do while loop

`do while` is an exit control loop where first loop body executes and then condition is checked at the exit point.

Syntax:

```
do {
    //loop body
    // increment/ decrement
} while (condition);
```

Example:

```
let i = 1;

do {
    console.log(i);
    i++;
} while (i <= 10);
```

break keyword

`break` is a keyword which is used to terminate the execution of loop. When `break` keyword will execute then immediately that point loop execution will be suspended.

Example:

```
let i = 1;

while (true) {
    console.log(i);

    if (i == 10)
        break;
    i++;
}
```

continue keyword

`continue` statement is used to one or multiple executions from the loop. When `continue` executes then control immediately transferred to the increment statement.

⚠ Critical Behavior Differences:

☑ `for` loop - SAFE with `continue`

- When `continue` executes, control jumps directly to the **increment/update expression**, then re-checks the condition
- Since the increment is part of the loop structure, there's **no risk of infinite loop** from using `continue`

⚠ `while` loop - DANGEROUS with `continue`

- When `continue` executes, it skips remaining code and jumps directly to the **condition check**
- There is **no automatic increment**, so if you don't manually update the loop variable before `continue`, it causes an **infinite loop**

⚠ `do-while` loop - DANGEROUS with `continue`

- When `continue` executes, it skips remaining code and jumps directly to the **condition check** (similar to `while`)
- Again, there is **no automatic increment**, so if the loop variable isn't updated before `continue`, the loop becomes **infinite**

Safe Example (for loop):

```
for (let i = 0; i < 15; i++) {

    if (i == 7 || i == 10)
        continue;  // Safe - jumps to i++

    console.log(i);
}
```

Dangerous Example (while loop):

```
let i = 0;
while (i < 15) {

    if (i == 7)
        continue;  // INFINITE LOOP! i never increments

    console.log(i);
    i++;  // This line never executes when continue is triggered
}
```

for loop

- `for` loop is an entry control loop. which executes repeatdely until loop condition will be true.

Syntax

```
for (initialization; condition; increment / decrement){
    //loop body
}
```