Lesson:

For Loop, Break and Continue

PW SKILLS

Topics Covered

1. For loop
2. Nested For loop
3. Break Statement
4. Continue Statement

Loops are used in programming to repeatedly run a block of code.

Several loops in JS.

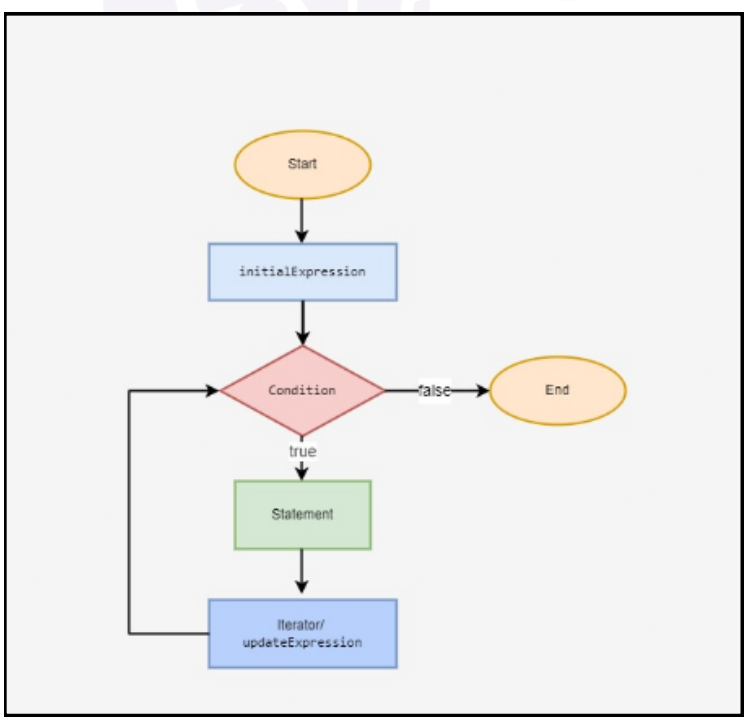
- for loop
- while loop
- do while loop

In this lesson, we will study about the **for loop.**

For Loop

```JavaScript
//Syntax

for (initial expression; condition; update expression) {
    // for loop body
}
```

Understanding flow



1. **InitialExpression** only ever executes once while initializing and declaring variables.
2. The **condition** is assessed.
   a. If the condition is **false,** the for loop will end.
   b. If the condition is true,
      i. The for loop's code block will be executed
      ii. **Update expression** changes the b value.
   c. The condition is once more assessed and the loop continues until the condition becomes false.

Example 1: Using For loop to print **"PW Skills"** 3 times.

```javascript
for (let i = 0; i < 3; i++)


{
    let name = "PW Skills";
    console.log(name);
}

// Output
/*
PW Skills
PW Skills
PW Skills
*/
```

Example 2: Display a sequence of even numbers till 20

```javascript
for (let i = 2; i <= 20; i+=2) {
    console.log(i);
}

// Output
/*
2
4
8
10
12
14
16
18
20
*/
```

Nested for loop

An outer for loop and one or more inside for loops constitute a nested for loop. Control re-enters the inner for loop and initiates a fresh execution every time the outer for loop repeats.

In other words, every time the outer for loop repeats, the control will enter the inner for loop.

Nested for loop often looks like this:

```javascript
// Outer for loop.
 for ( initialization; test-condition; increment/decrement )
 {
// Inner for loop.
   for ( initialization; test-condition; increment/decrement )
   {
    // statement of inner loop
   }
 // statement of outer loop
}
```

The execution flow is something like this:

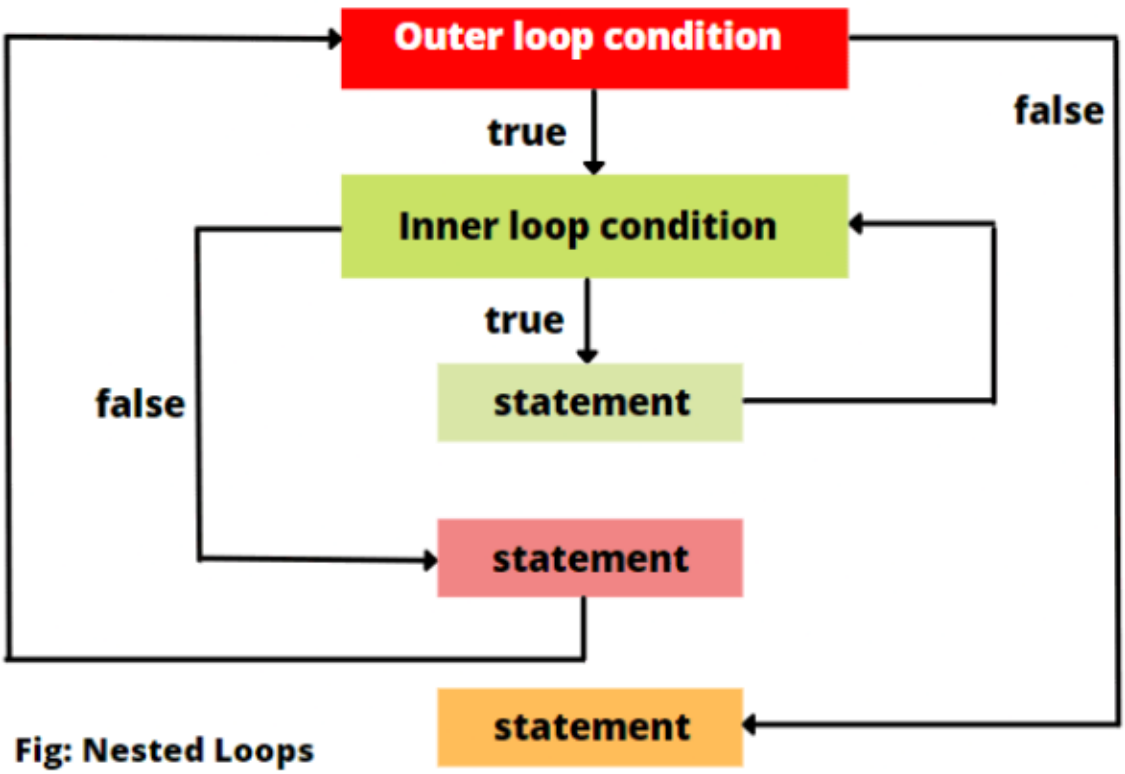


Fig: Nested Loops

Example 1: Write a program to show the inner for loop values for each outer iteration along with the inner for loop.

```javascript
for(let i=1;i<=3;i++){ //outer loop
    console.log("for i= " + i + " the innerloop values are")
    for(let j=1;j<3;j++){  //inner loop
        console.log("j= "+j)
    }
}

// Output

/*
for i=1 the innerloop values are
j=1
j=2
for i=2 the innerloop values are
j=1
j=2
for i=3 the innerloop values are

j=1
j=2
*/
```

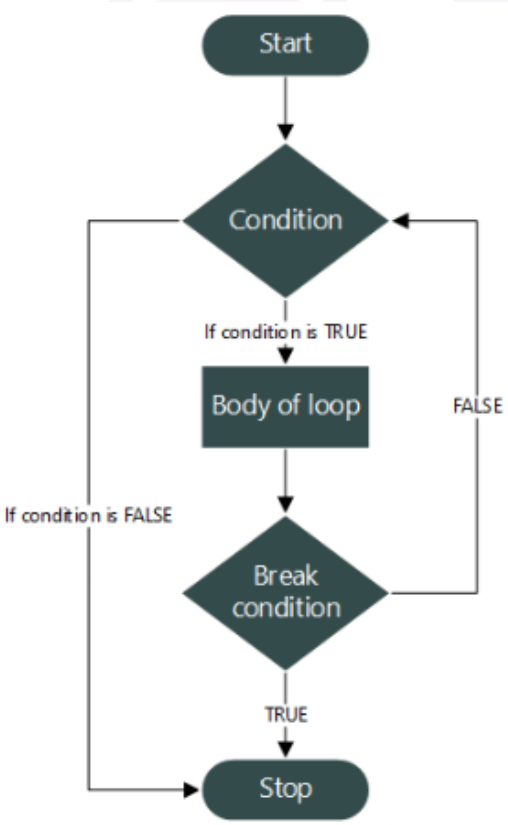
Example 2: Write a program to draw following pattern,

```
*
* *
* * *
* * * *
* * * * *
* * * *
* * *
* *
*
```

```javascript
for (let i = 1; i <= 5; i++) {
    let row = "";
    for (let j = 1; j <= i; j++) {
      row += "* ";
    }
    console.log(row);
  }

  for (let i = 4; i >= 1; i--) {
    let row = "";
    for (let j = 1; j <= i; j++) {
      row += "* ";
    }
    console.log(row);
  }
```

Break Statement

We can terminate the loop explicitly using the break statement.



Example of using break with for loop:

Can you guess the output of the following code ?

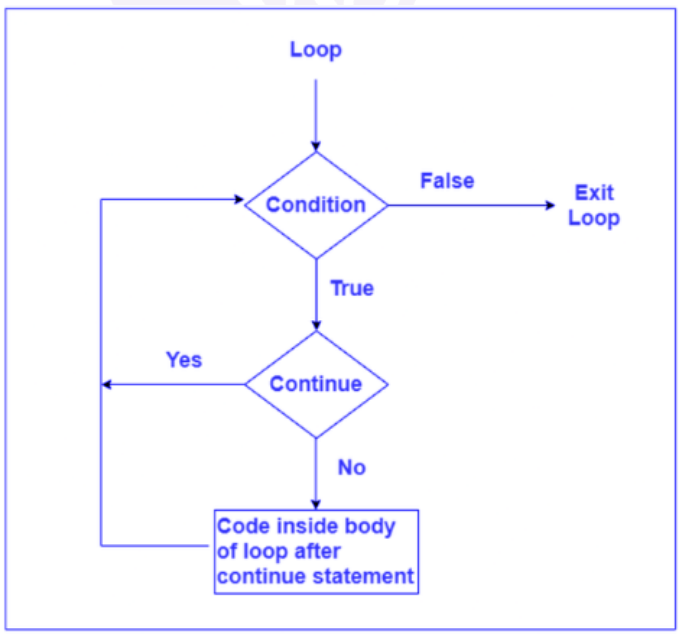
```javascript
for (let i = 0; i < 4; i++) {
  console.log(i);
  if (i == 2) {
    break;
  }
}

/* Output
0
1
2
*/
```

Here, we use an **if statement** inside the loop. If the current value of i becomes 2, the if statement will execute and the ***break* statement** will terminate the loop.

That is why we only see numbers till 2 in the output.

Continue Statement

Continue statement skips the current iteration of loop, and moves on to the subsequent iteration.



Example of using continue in for loop
Write a program to display only odd numbers till 20:

```javascript
for (let i = 0; i < 20; i++) {
   if (i % 2 === 0) {
     continue;
   }
   console.log(i);
}

/* Output
1
3
5
7
9
11
13
15
17
19
*/
```

Here, the for loop iterates through the values from 0 to 20.

The remainder of the division of the current value of i by 2 is returned by the i%2 expression.

The **continue** statement, which skips the current iteration of the loop and moves to the iterator expression i++, is executed if the remainder is zero. If not, the value of i is output to the console.

Interview Point.

1. Pattern Printing: Create a program that prints a right-angled triangle pattern of stars. Consider the height to be 5.

Ans:

```
// Function to print a right-angled triangle pattern of stars
function printRightAngleTriangleStars(height) {
    for (let i = 1; i <= height; i++) {
        // String to store the stars for each row
        let row = "";

        // Concatenate stars for each row
        for (let j = 1; j <= i; j++) {
            row += "* ";
        }

        // Print the row using console.log
        console.log(row);
    }
}

// Call the function with a height of 5
printRightAngleTriangleStars(5);
```

2. What are loop labels and explain their usage with an example?

Ans: Loop labels, also known as loop identifiers or loop names, are used to provide a way to control the flow of nested loops.

A loop label is an identifier followed by a colon (:) placed before a loop statement. This label allows you to reference the loop by name and use it with control flow statements like break and continue to specify which loop to affect.

```
outer: for (let i = 1; i <= 3; i++) {
    for (let j = 1; j <= 3; j++) {
        if (i > 2) break outer;
        console.log(i, j);
    }
}
```

In the above example, the purpose of the **outer** label is to break out of both the outer and inner loops when the condition if (i > 2) is met. Without the outer label, the break statement would only exit the inner loop where it is located.

Here's a step-by-step explanation:

1. The outer loop (for (let i = 1; i <= 3; i++)) is labeled with outer.
2. Inside the outer loop, there's an inner loop (for (let j = 1; j <= 3; j++)).
3. If the condition if (i > 2) is true (when i is greater than 2), the break outer; statement is executed.

4. When the break outer; statement is executed, it breaks out of both the inner and outer loops, terminating the entire loop structure.

In other words, the outer label allows the break statement to target the outer loop for termination when a specific condition is met within the inner loop. It provides a way to control the flow of both loops simultaneously.