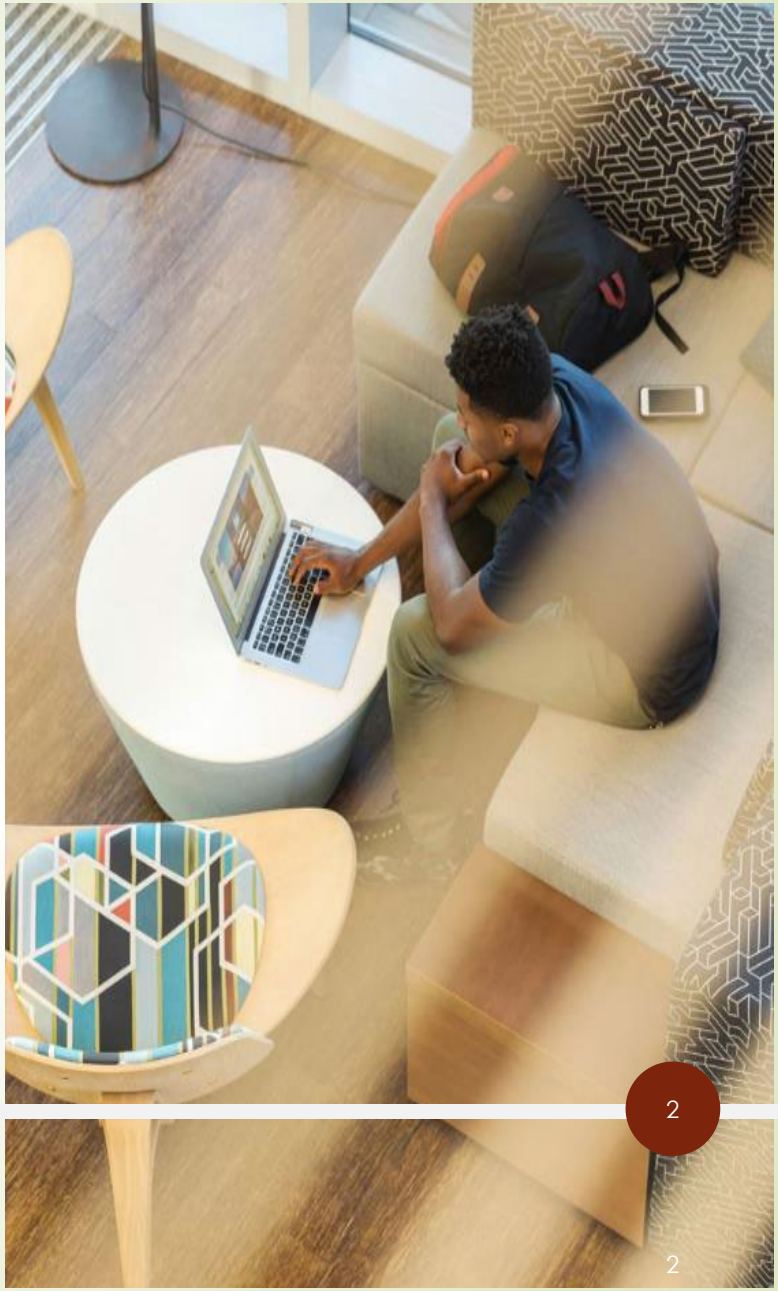
ST0523 Fundamentals of Programming

Topic 4: Repetitions (do-while & while loops)

Topic 4: Repetitions (do-while & while loops)

- To use do-while and while loop statements to control the repetition of statements.
- To know the similarities and differences of three types of loops.
- To write nested loops.



2

Repetitions

- A more efficient way is to make use of repetition/ looping/iteration structures.
  - for Loop
  - while Loop
  - do-while Loop
- We'll be focusing on **do-while** & **while** loops for this topic.

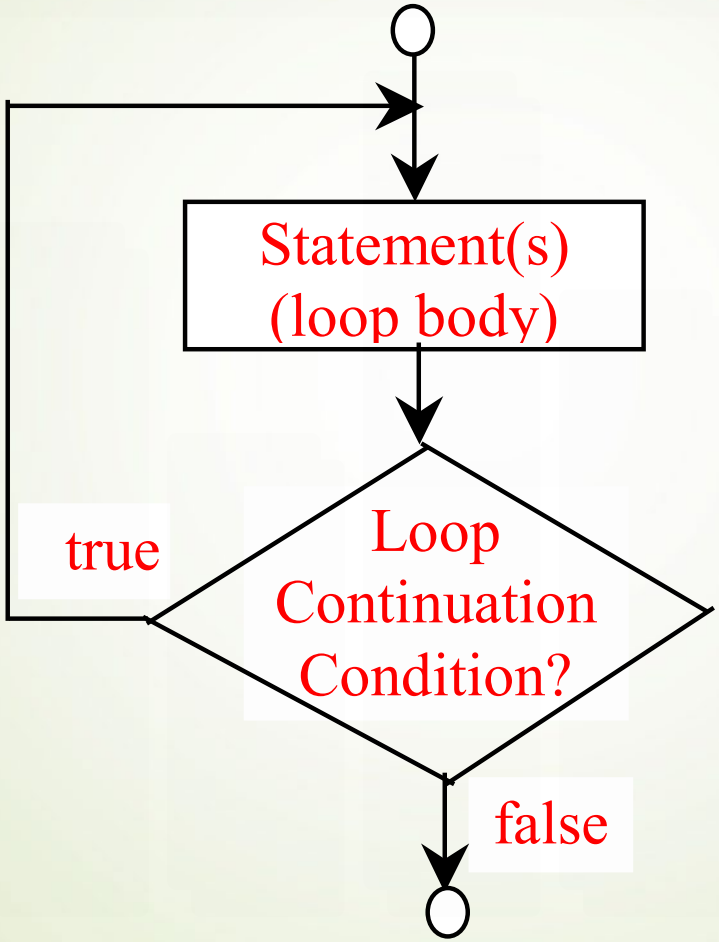
do-while Loop Syntax

```
do {
    // loop body;
    Statement(s);
} while (condition);
```

Example :

```
var count = 0;
do {
      console.log("Welcome to FOP!");
      count++;
} while (count < 10);
```

Aided Flowchart



How **do-while** Loop Works

- Loop body is first executed followed by a test on the loop continuation condition (boolean expression).
- If the condition evaluates to **true**, execute loop body again.
- Repeat process until the condition evaluates to **false**.
- The loop body is executed at least once.
- Thus, *do-while loop* is designed for solving problems in which **at least one iteration** must occur.

do-while Loops Example

```
var counter = 0;
do {
     console.log(counter);
     counter++;
} while (counter <= 2);
console.log("Done");
```

Output:

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter < 2);
console.log("Done");
```

Output:

Declare counter and initialize to 0

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

Output:

Proceed within the brackets { }

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
```

Print 0

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
```

Execute adjustment statement
counter now is 1

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
```

(counter <= 2) is true
since counter is 1

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
```

Print 1

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
```

Execute adjustment statement
counter now is 2

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
```

(counter <= 2) is true
since counter is 2

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
2
```

Print 2

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
2
```

Execute adjustment statement
counter now is 3

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
2
```

(counter <= 2) is false since counter is 3.
Hence exiting the loop.

do-while Loops Example

```
var counter = 0;
do {
    console.log(counter);
    counter++;
} while (counter <= 2);
console.log("Done");
```

```
Output:
0
1
2
Done
```

Print Done

Example: Using **do-while** Loop

- Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

Example: Using **do-while** Loop

```
var input = require('readline-sync');
var data;
var sum = 0;

// Keep reading data until the input is 0

 do {

   // Read the next data
   var dataString = input.question(
    "Enter an int value (the program exits if the input is 0): ");

   data = parseInt(dataString);
    sum += data;

  } while (data != 0);

  console.log("The sum is " + sum);
```

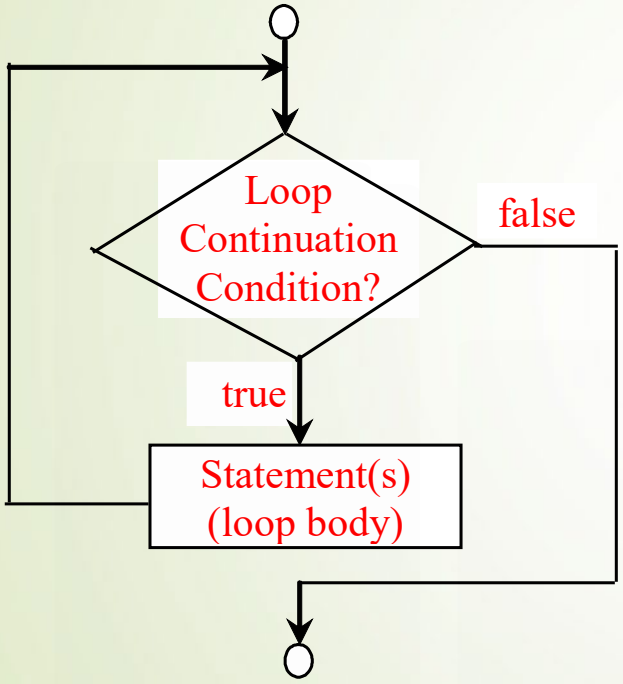
while Loop Syntax

```
while (condition) {
    // loop body;
    Statement(s);
}
```

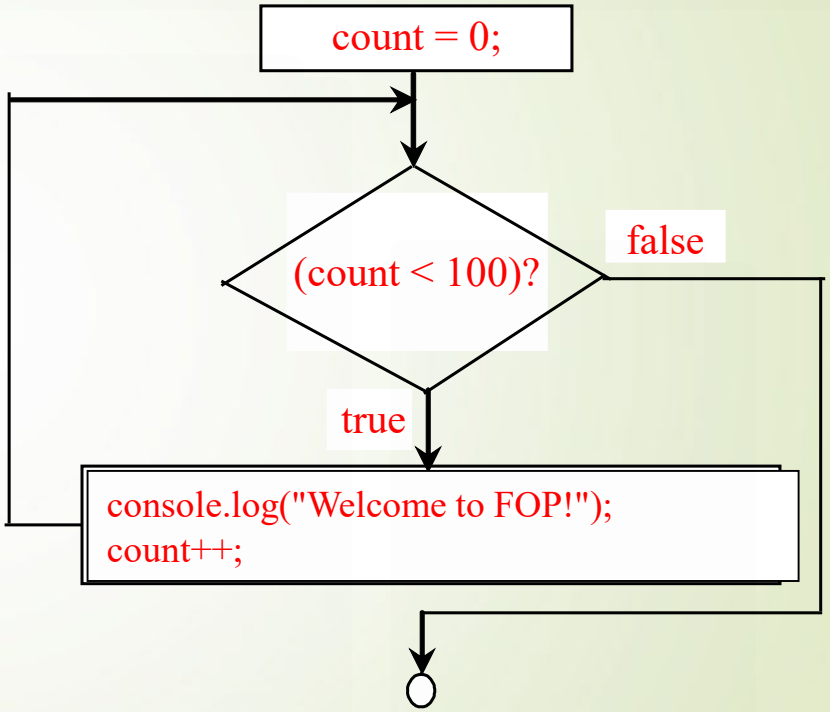
Example

```
var count = 0;
while (count < 100) {
  console.log("Welcome to FOP!");
  count++;
}
```

Aided Flowchart



(A)



(B)

How **while** Loop Works

- The while loop begins by evaluating the Boolean expression.
- If the expression evaluates to **true**, the statement(s) are executed.
- Control then returns to evaluate the Boolean expression again.
- This process repeats itself until the Boolean expression evaluates to **false**.

while Loops Example

```
var number = 1;
while (number < 3) {
     console.log(number);
     number++;
}
console.log("Done");
```

Output:

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done
```

Output:

Declare number and initialize to 1

while Loops Example

```
var number = 1;
while (number < 3) {
      console.log(number);
      number++;
}
console.log("Done"
```

Output:

(number < 3) is true since number is 1

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
```

Print 1

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
```

Execute adjustment statement
number now is 2

while Loops Example

```
var number = 1;
while (number < 3) {
      console.log(number);
      number++;
}
console.log("Done"
```

```
Output:
1
```

(number < 3) is true since number is 2

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
2
```

Print 2

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
2
```

Execute adjustment statement
number now is 3

while Loops Example

```
var number = 1;
while (number < 3) {
      console.log(number);
      number++;
}
console.log("Done"
```

```
Output:
1
2
```

(number < 3) is false since number is 3

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
2
```

Exit the loop. Execute the next statement after the loop

while Loops Example

```
var number = 1;
while (number < 3) {
    console.log(number);
    number++;
}
console.log("Done");
```

```
Output:
1
2
Done
```

Print "Done"

Example: Using **while** Loop

- Write a program that reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.

Example: Using **while** Loop

```
var input = require('readline-sync');

// Read an initial data
var dataString = input.question(
  "Enter an int value (the program exits if the input is 0): ");

var data = parseInt(dataString);

// Keep reading data until the input is 0
var sum = 0;
while (data != 0) {
  sum += data;

// Read the next data
dataString = input.question(
  "Enter an int value (the program exits if the input is 0) :");
  data = parseInt(dataString);
}
```

Which Loop to use?

- The three forms of loop statements, while, do-while, and for, are expressively equivalent; that is, you can write a loop in any of these three forms.
- For example, a while loop in (A) in the following figure can always be converted into the following for loop in (B):

```
while (loop-continuation-condition) {
  // Loop body
}
```

(A)

Equivalent

```
for ( ; loop-continuation-condition; )
  // Loop body
}
```

(B)

Which Loop to use?

- A for loop in (A) in the following figure can generally be converted into the following while loop in (B):

```
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

(A)

Equivalent

```
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```

(B)

Which Loop to use?

- Use **for** loop if the number of repetitions is known, e.g. when you need to print a message 100 times.
- Use **while** loop if the number of repetitions is not known, e.g. reading the numbers until the input is 0.
- Use **do-while** loop to replace a while loop if the loop body has to be executed before testing the continuation condition.

Caution

- The following **while** loop is also wrong:

```
var i=0;
while (i < 10);
{
        console.log( "i is " + i );
        i++;
}
```

Logic Error

Caution

- In the case of the **do-while** loop, the following semicolon is needed to end the loop.

```
var i=0;
do {
      console.log( "i is " + i );
      i++;
} while (i<10);
```

Correct

Class Exercise

- Convert the following **for** loop statement to a **while** loop and to a **do-while** loop:

```
var sum = 0;
for (var  i = 0;  i <= 1000;  i++)  {
        sum =  sum  +  i;
}
```

How to run a nested for loop

Step by step guide

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

Output:

Declare x and initialize to 0

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log( + y);
      }
      console.log("*");
}
```

```
Output:
```

(x < 2) is true since x is 0

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
```

Print "*"

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
```

Declare y and initialize to 3

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
```

(y > 1) is true since y is 3

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
```

Since x = 0 & y = 3,
x + y = 0 + 3 = 3
Print 3

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
```

Execute adjustment statement
y now is 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
```

(y > 1) is true since y is 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
```

Since x = 0 & y = 2,
x + y = 0 + 2 = 2
Print 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
```

Execute adjustment statement
y now is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
```

(y > 1) is false since y is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
```

Exit the loop. Execute the next statement after the loop

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
```

Print "*"

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
```

Execute adjustment statement
x now is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log( + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
```

(x < 2) is true since x is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
```

Print "*"

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
```

Declare y and initialize to 3 AGAIN

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
```

(y > 1) is true since y is 3

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
```

Since x = 1 & y = 3,
x + y = 1 + 3 = 4
Print 4

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
```

Execute adjustment statement
y now is 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
```

(y > 1) is true since y is 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
```

Since x = 1 & y = 2,
x + y = 1 + 2 = 3
Print 3

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
```

Execute adjustment statement
y now is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
```

(y > 1) is false since y is 1

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
```

Exit the loop. Execute the next statement after the loop

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
*
```

Print "*“ AGAIN

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
*
```

Execute adjustment statement
x now is 2

Nested for Loops Example

```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log( + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
*
```

(x < 2) is false since x is 2

Nested for Loops Example

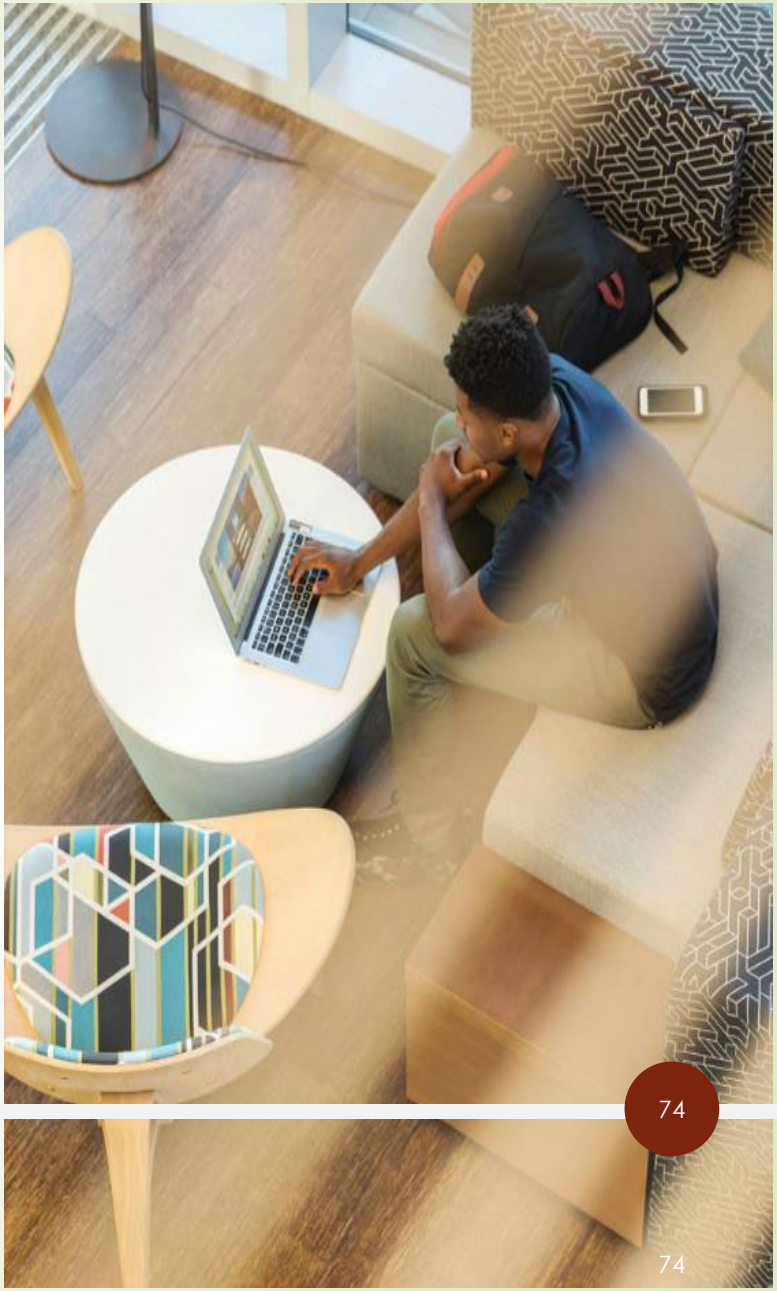
```
for (var x = 0; x < 2; x++) {
      console.log("*");
      for (var y = 3; y > 1; y--) {
            console.log(x + y);
      }
      console.log("*");
}
```

```
Output:
*
3
2
*
*
4
3
*
```

Exit the loop. Execute the next statement after the loop

Summary for Topic 4: Repetitions (do-while & while loops)

- To use do-while and while loop statements to control the repetition of statements.
- To know the similarities and differences of three types of loops.
- To write nested loops.



74

SP SCHOOL OF Computing