Problem Solving

LOOPS

SA, 11 DEC 2021

Why use loops?

Suppose you want to print the numbers from 1 to 100.

With being able to copy and paste, you can do something like this:

```
cout<<1<<endl;
cout<<2<<endl;
cout<<3<<endl;
cout<<4<<endl;
cout<<5<<endl;
cout<<6<<endl;
```

and so on until you get to `cout<<100<<endl;` As you can see, this code could take a long time to type `(Ctrl + C and Ctrl + V)` so there is **the Loops**.

Loops

Loops are used in programming when we need to repeatedly execute a set of instructions. Like the previous example for printing. That is, a loop is a series of instructions that are repeated until a certain state is reached. There are two types of loops:

- Entry Controlled loops:

  In this type of loops the test condition is tested before entering the loop body. `For Loop` and `While Loop` are entry controlled loops.

- Exit Controlled Loops:

  In this type of loops the test condition is tested or evaluated at the end of loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. `do – while loop` is exit controlled loop.

Loops

For loop

```
for(int i=0; i<10 ; i++) {}
```

- Use a for loop when you have a counter variable and you want it to loop while the counter variable increases or decreases over a range.

While loop

```
while(i < 10) {}
```

- Use the while loop when you have a condition under which you want your code to run

Do - while

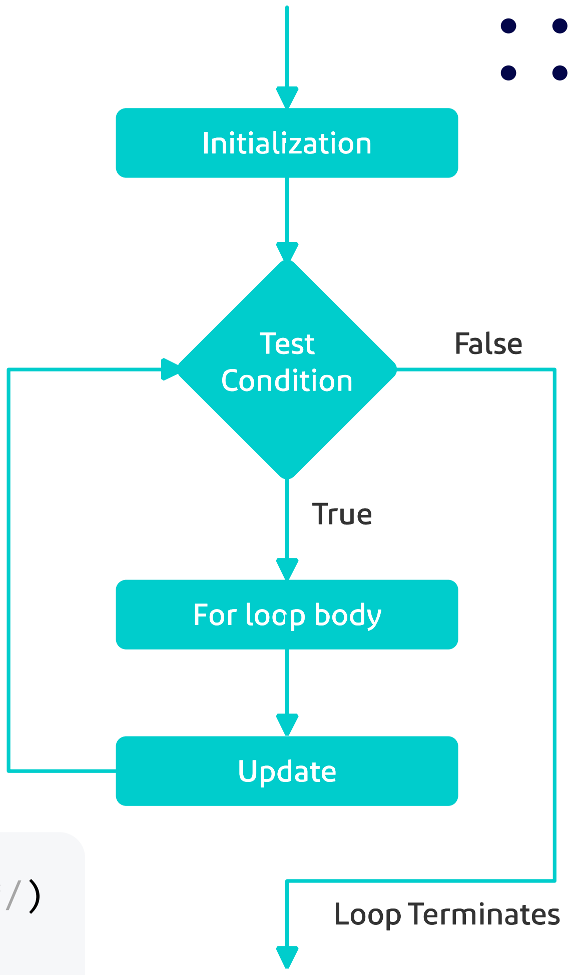
```
do {} while(i < 10);
```

- Use it when you have a condition that you want your code to run and you want to make sure that the loop runs at least once, even if the condition is not met.

For Loop

- The for loop is used to execute a set of statements repeatedly until a particular condition is satisfied.
- To use a `for loop`, you use the `for` keyword and follow it with a set of parentheses that contains information regarding the number of times the `for loop` executes.

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



For Loop

This statement gets executed only once, at the beginning of the `for` loop. You can enter a declaration of multiple variables of one type, such as `int i = 0, a = 2, b = 3;`. These variables are only valid in the scope of the loop.

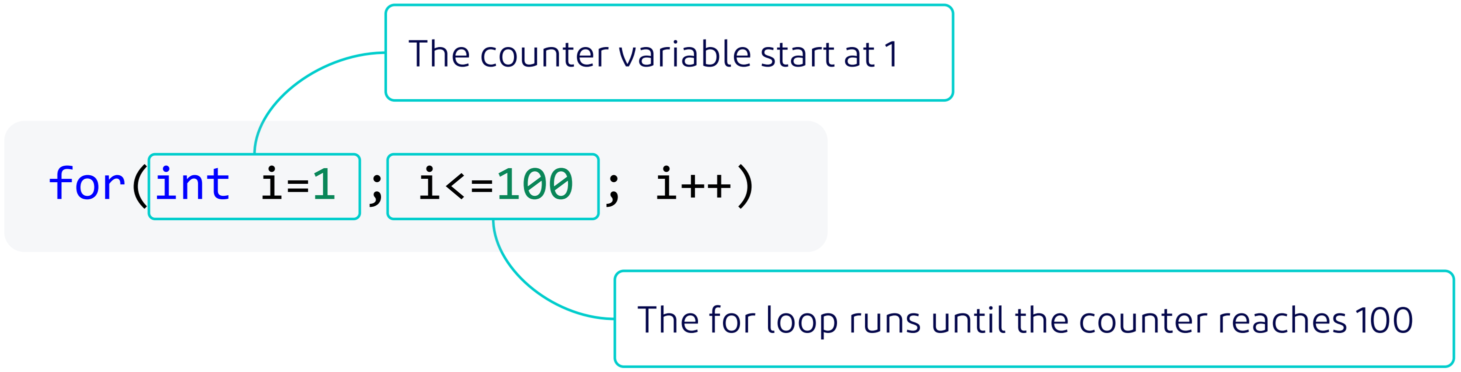
```
for (/*initialization*/ ; /*condition*/ ; /*increment/decrement*/)
{
    // body of the loop
}
```

This statement gets evaluated ahead of each loop body execution, and end the loop if it evaluates to false

This statement gets executed after the loop body, ahead of the next condition evaluation, unless the for loop is aborted in the body (by `break, goto, return`).

For Loop

- When printing numbers 1 to 100, you want a variable that starts with `1;` Then you print the number `1,` increment the variable to `2,` and print the next number over and over.
- The common procedure here that doesn't change every time is the "print the number" part, and the part that changes is the variable, called the counter variable.
- The `for` statement looks like this:



```
for(int i=1; i<=100; i++)
```

The counter variable start at 1

The for loop runs until the counter reaches 100

For Loop

- Now, to print the numbers, you can do this:

```cpp
#include <iostream>
using namespace std;

int main()
{
    for(int i=1 ; i<=100 ; i++)
    {
        cout<<i<<" ";
    }

    return 0;
}
```

```
Output: 1 2 3 4 5 ... 95 96 97 98 99 100
```

Iteration	Variable	`i <= 100`	Actions
1	`i = 1`	`True`	`1` is printed. `i` is increased to `2`.
2	`i = 2`	`True`	`2` is printed. `i` is increased to `3`.
3	`i = 3`	`True`	`3` is printed. `i` is increased to `4`.
		. . . . .	
99	`i = 99`	`True`	`99` is printed. `i` is increased to `100`.
100	`i = 100`	`True`	`100` is printed. `i` is increased to `101`.
101	`i = 101`	`False`	The loop terminated

For Loop

- If you need multiple counter variables, the for loop can handle it. Each portion of the for statement can have multiple items in it, separated by commas.
- For example, the following line of code uses two counter variables:

Here, the code creates two counters `i` and `j`. `i` starts at `1`, and `j` starts at `100`.

`i` is incremented by `1`, and `j` is decremented by `1`.

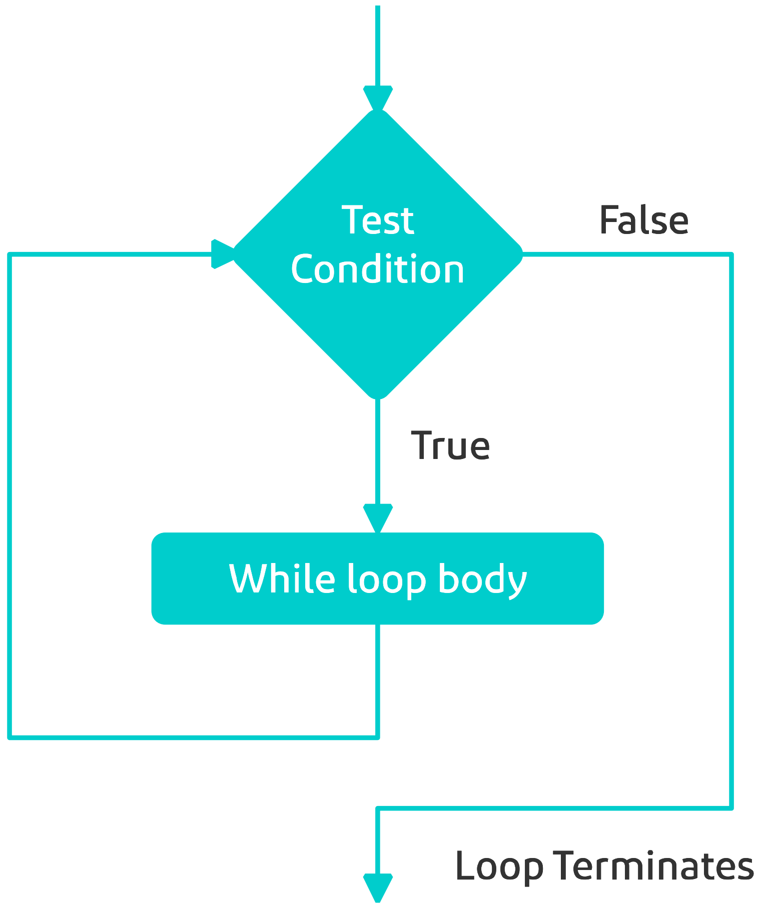
```
for(int i=1, j=100 ; i<=100 && j>0 ; i++, j--)
{
    cout<<i<<" "<<j<<endl;
}
```

The loop will run as long as the following two conditions are `true`: `i` must be less than or equal to `100`, and `j` must be greater than `0`.

While Loop

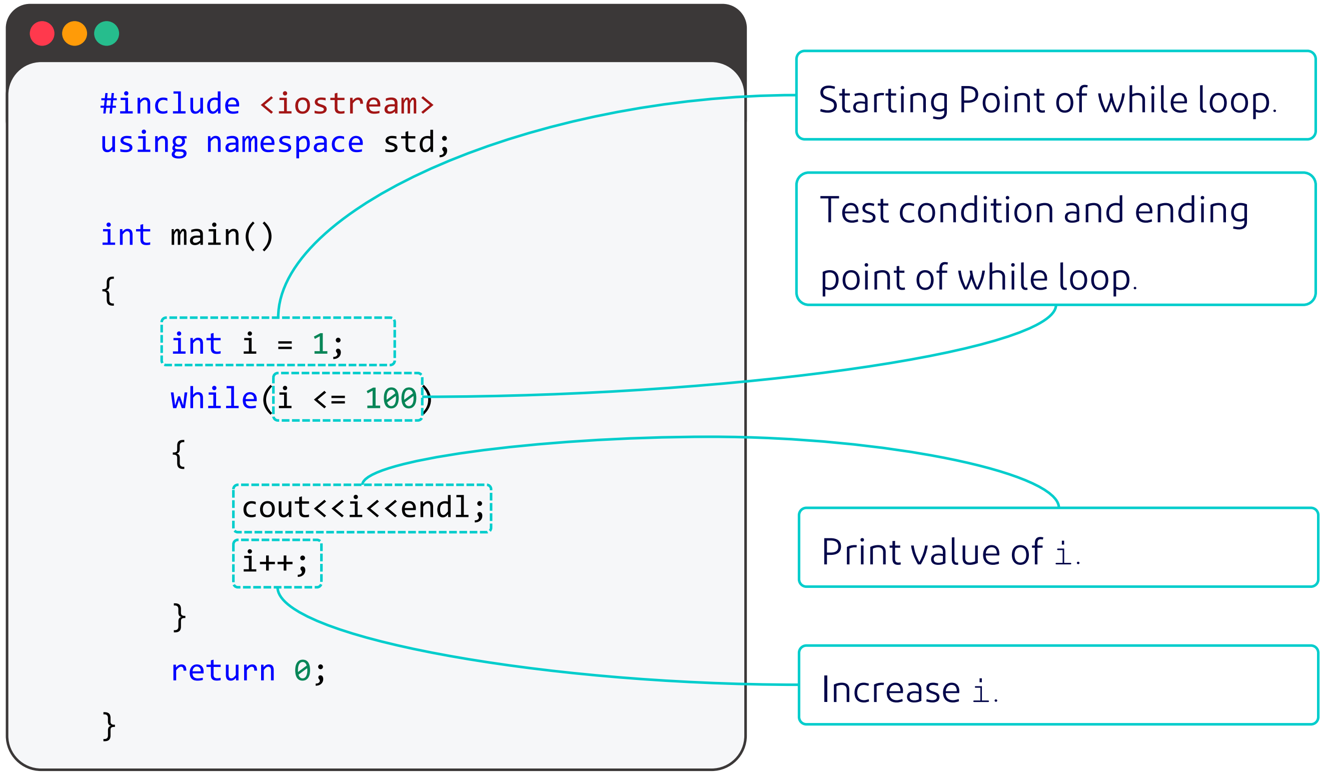
- Often, you find that for loops work only so well. Sometimes, you don’t want a counter variable; you just want to run a loop over and over as long as a certain situation is true. Then, after that situation is no longer the case, you want to stop the loop to do this in C++, you use a while statement.

```
while (/*condition*/)
{
    // body of the loop
}
```



While Loop

- The while loop example to print numbers from 1 to 100

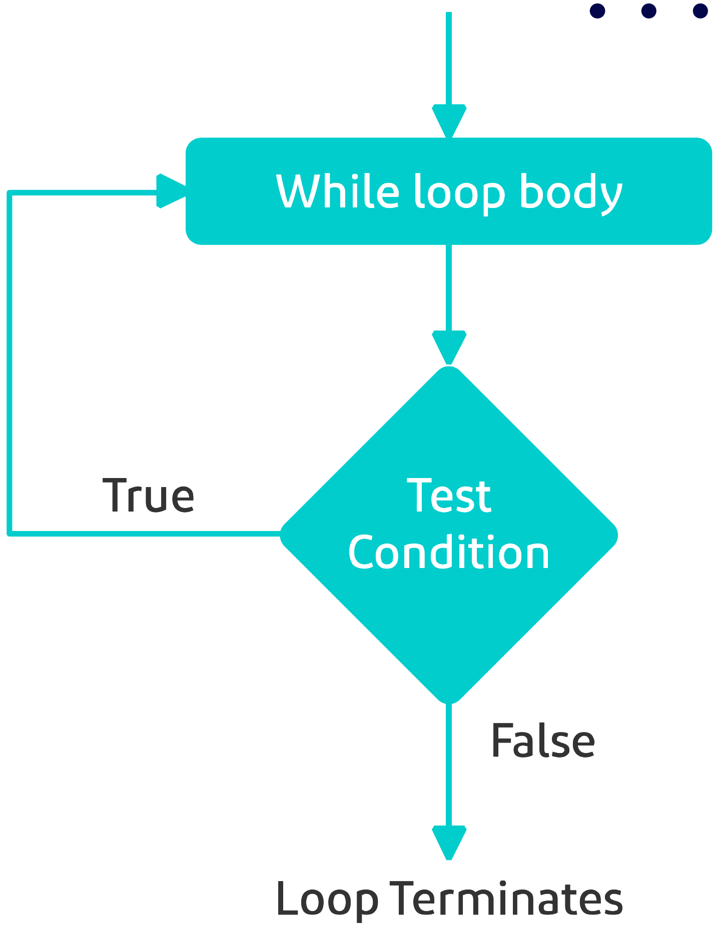


```
#include <iostream>
using namespace std;

int main()
{
    int i = 1;
    while(i <= 100)
    {
        cout<<i<<endl;
        i++;
    }
    return 0;
}
```

Starting Point of while loop.
Test condition and ending point of while loop.
Print value of i.
Increase i.

Do While Loop

- A `do-while` loop is very similar to a while loop, except that the condition is checked at the end of each cycle, not at the start.
- The `do-while` loop has one important caveat: Unlike the `while` loop, the `do-while` loop always runs at least once. In other words, even if the condition isn't satisfied the first time you run the loop, it runs anyway.



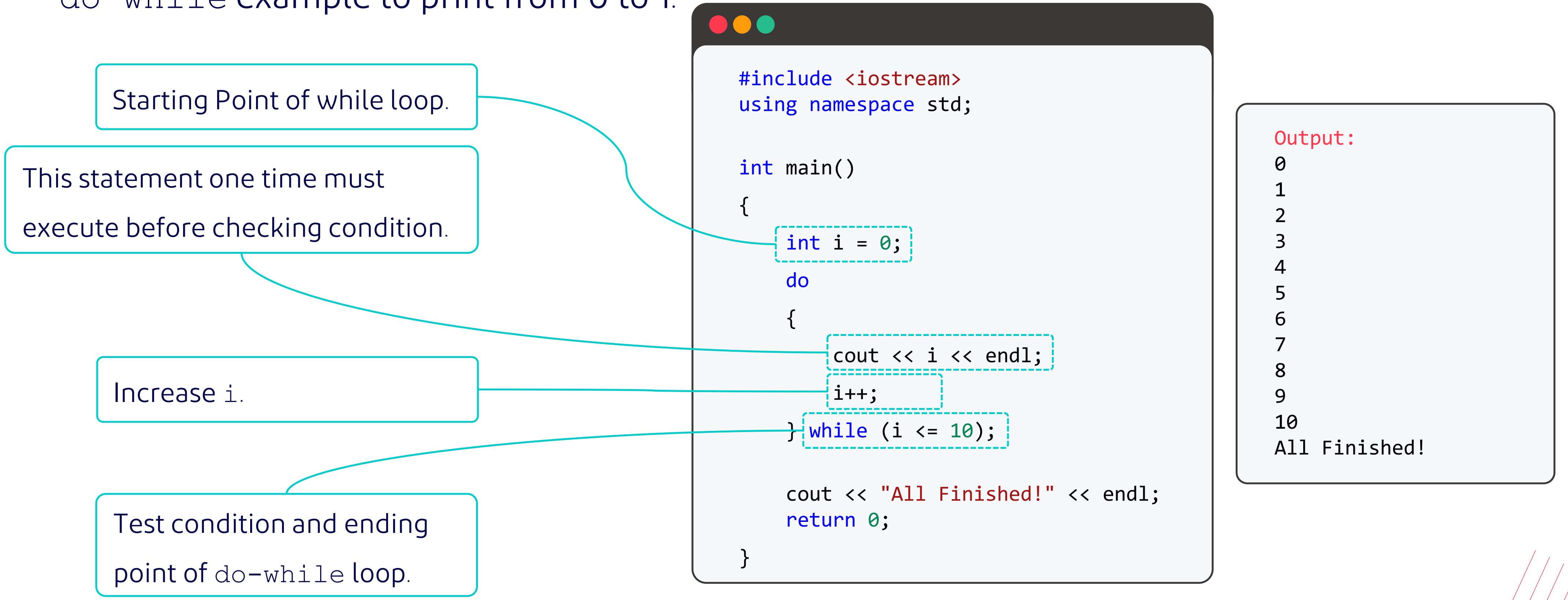
Do While Loop

- The test condition must be enclosed in parentheses and followed by a semi-colon.
- The `do-while` loop is an exit this means that the body of the loop is always executed first. Then, the test condition is evaluated.
    - If the condition is `true`, the program executes the body of the loop again.
    - If the condition is `false`, the loop terminates and program continues with the statement following the `while`.

```
do {
    // body of the loop
}
while (/*condition*/);
```

Do While Loop

- `do-while` example to print from 0 to 1.



Starting Point of while loop.

This statement one time must execute before checking condition.

Increase `i`.

Test condition and ending point of `do-while` loop.

```cpp
#include <iostream>
using namespace std;

int main()
{
    int i = 0;
    do
    {
        cout << i << endl;
        i++;
    } while (i <= 10);

    cout << "All Finished!" << endl;
    return 0;
}
```

```
Output:
0
1
2
3
4
5
6
7
8
9
10
All Finished!
```

Do While Loop

- Difference between `do–while` and `while` loop:

while

- It is entry controlled loop
- The loop executes the statement only after testing condition.
- There is no semicolon at the end of while statement.

do-while

- It is exit controlled loop
- The loop executes the statement at least once
- There is semicolon at the end of while statement.

Break and continue

Sometimes, you may write an application that includes a loop that does more than simply add numbers. You may find that you want the loop to end under a certain condition that's separate. Or you may want the loop to suddenly skip out of the current loop and continue with the next item in the loop.

- When you stop a loop and continue with the code after the loop, you use a `break` statement.
- When you quit the current cycle of the loop and continue with the next cycle, you use a `continue` statement.

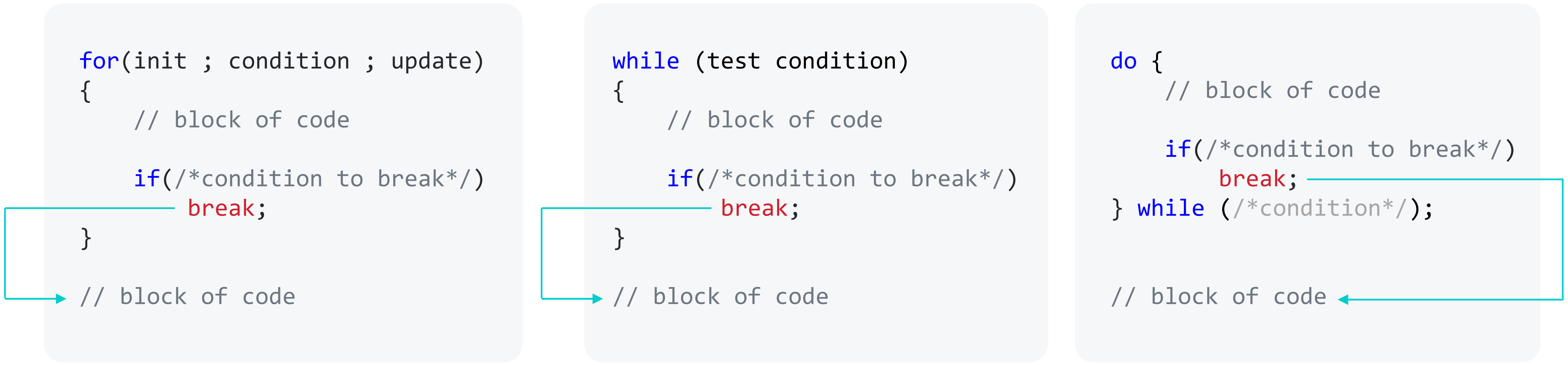
Break

- The `break` statement terminates a loop (`for`, `while` and `do-while` loop).
- `break` statement stops the loops and start executing program from the line after loop.
- In real practice, `break` statement is almost always used inside the body of conditional statement (`if...else`) inside the loop.
- Suppose you're writing an app that prints multiples of 1 to 10, but when you hit 5, you want to exit the for loop immediately.

How to do this?

Break

- How `break` statement works?



```
for(init ; condition ; update)
{
    // block of code

    if(/*condition to break*/)
        break;
}

// block of code
```

```
while (test condition)
{
    // block of code

    if(/*condition to break*/)
        break;
}

// block of code
```

```
do {
    // block of code

    if(/*condition to break*/)
        break;
} while (/*condition*/);


// block of code
```

Break

- Suppose you're writing an app that prints multiples of 1 to 10, but when you hit 5, you want to exit the for loop immediately.

```
Output:
1 -> 2
2 -> 4
3 -> 6
4 -> 8
All Finished!
```

```
#include <iostream>

using namespace std;

int main()
{

    for(int i=1 ; i<=10 ; i++)
    {
        if(i == 5)
            break;

        cout<<i<<" -> "<<i * 2<<endl;
    }

    cout<<"All Finished!"<<endl;

    return 0;
}
```

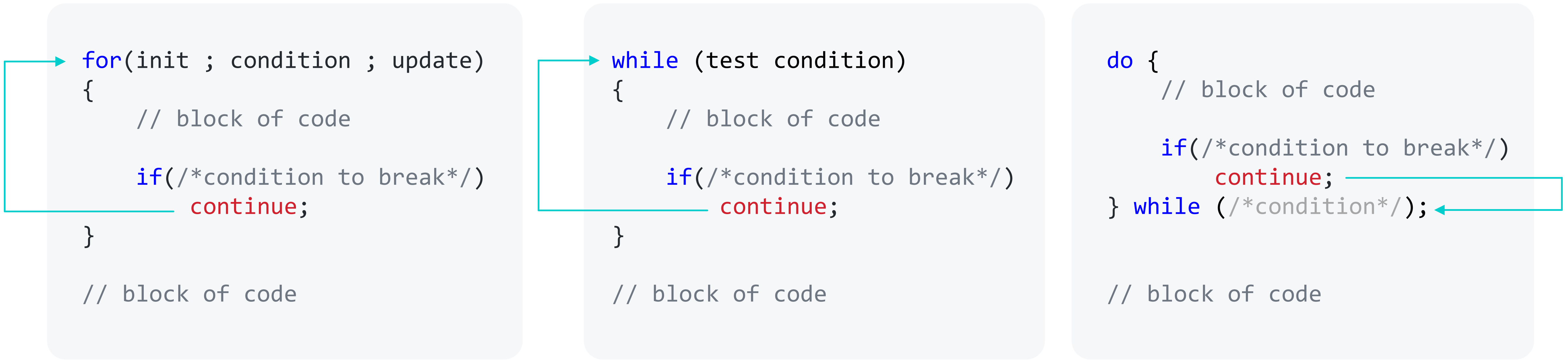
Continue

- In addition to the times when you may need to break out of a loop for a special situation, you can also cause the loop to end its current iteration; but instead of breaking out of it, the loop resumes with the next iteration.
- In real practice, `continue` statement is almost always used inside the body of conditional statement (`if`…`else`) inside the loop.
- Suppose you're writing an app that prints multiples of 1 to 10, except 5 and 7.

How to do this?

Continue

- How `Continue` statement works?



```
for(init ; condition ; update)
{
    // block of code

    if(/*condition to break*/)
        continue;
}

// block of code
```

```
while (test condition)
{
    // block of code

    if(/*condition to break*/)
        continue;
}

// block of code
```

```
do {
    // block of code

    if(/*condition to break*/)
        continue;
} while (/*condition*/);


// block of code
```

Continue

- Suppose you're writing an app that prints multiples of 1 to 10, except 5 and 7.

```
Output:
1 -> 2
2 -> 4
3 -> 6
4 -> 8
6 -> 12
8 -> 16
9 -> 18
10 -> 20
All Finished!
```

```cpp
#include <iostream>

using namespace std;

int main()
{

    for(int i=1 ; i<=10 ; i++)
    {
        if(i == 5 || i == 7)
            continue;

        cout<<i<<" -> "<<i * 2<<endl;
    }

    cout<<"All Finished!"<<endl;

    return 0;
}
```

Nested Loops

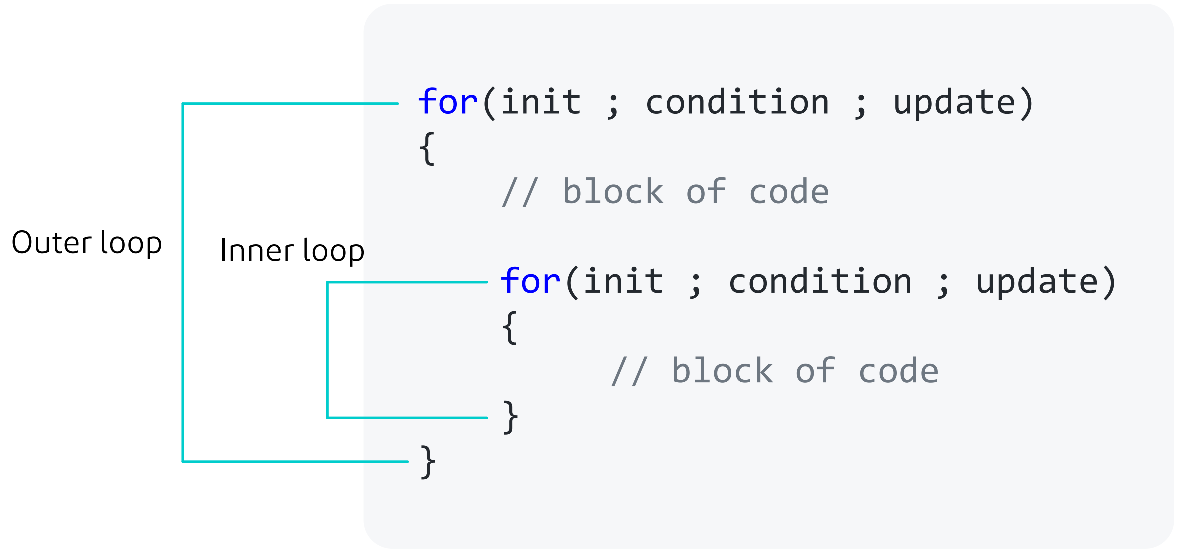
- A nested loop is simply a loop inside a loop
- There is no rule that a loop must be nested inside its own type. In fact, there can be any type of loop nested inside any type and to any level.
- Suppose you are writing a program that prints the numbers 1 to 10 five times.

How to do this?

Nested Loops

Nested loops are useful when for each pass through the outer loop, you need to repeat some action on the data in the outer loop.

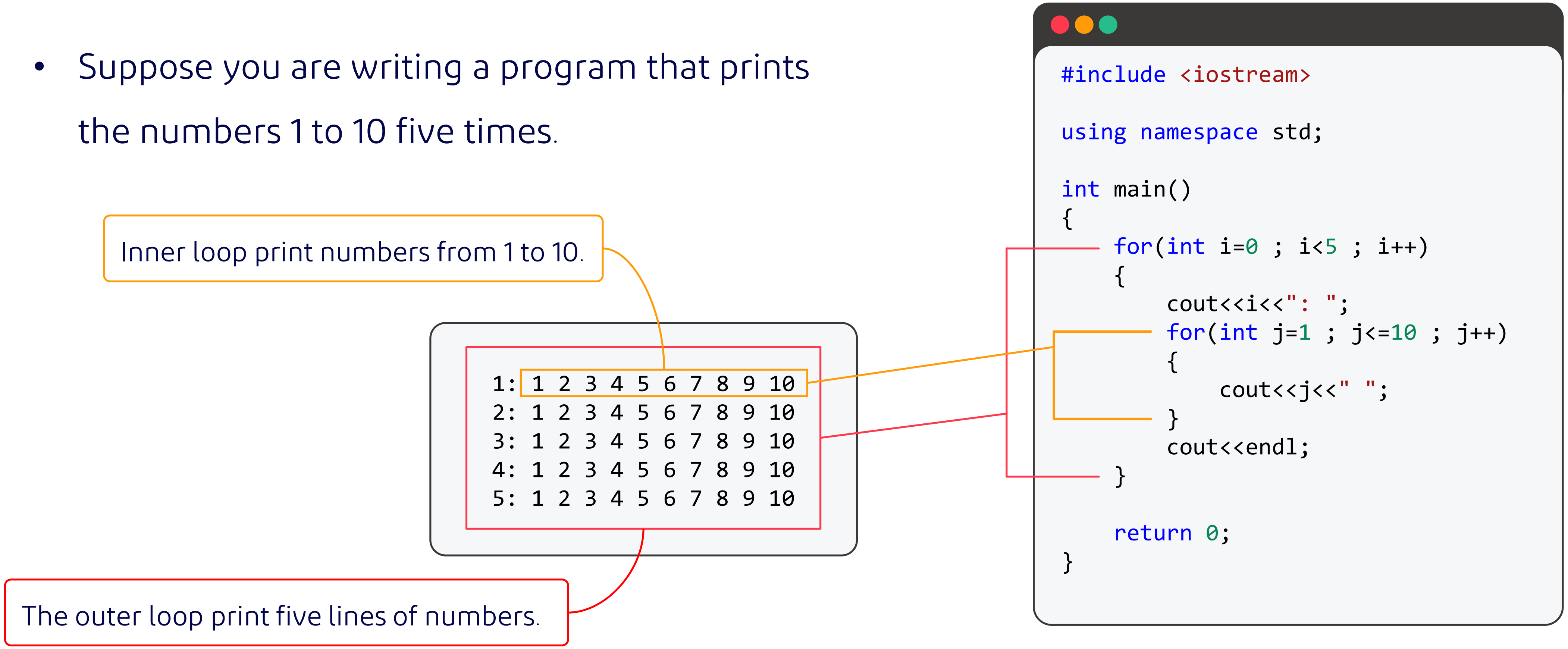
- First inner loop complete its iteration and then control shift to outer loop, this process continues till end.



```
for(init ; condition ; update)
{
    // block of code

    for(init ; condition ; update)
    {
        // block of code
    }
}
```

Nested Loops

- Suppose you are writing a program that prints the numbers 1 to 10 five times.



Inner loop print numbers from 1 to 10.

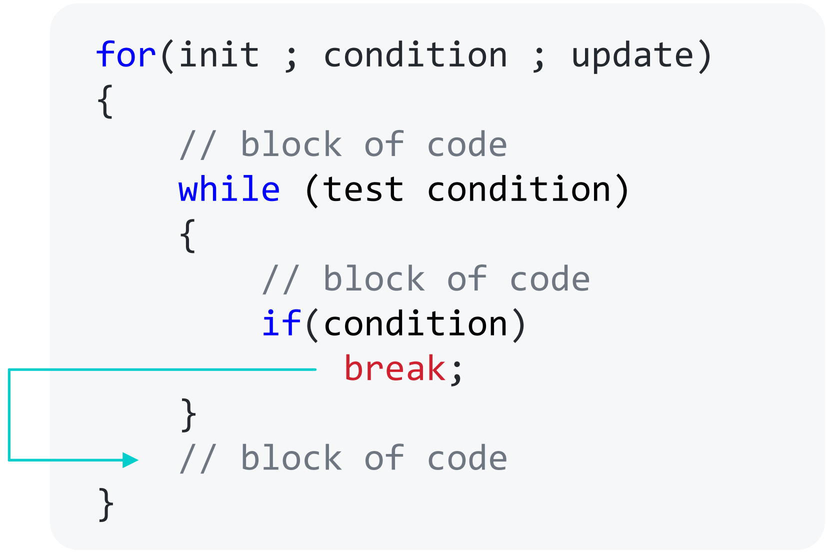
```
1: 1 2 3 4 5 6 7 8 9 10
2: 1 2 3 4 5 6 7 8 9 10
3: 1 2 3 4 5 6 7 8 9 10
4: 1 2 3 4 5 6 7 8 9 10
5: 1 2 3 4 5 6 7 8 9 10
```

The outer loop print five lines of numbers.

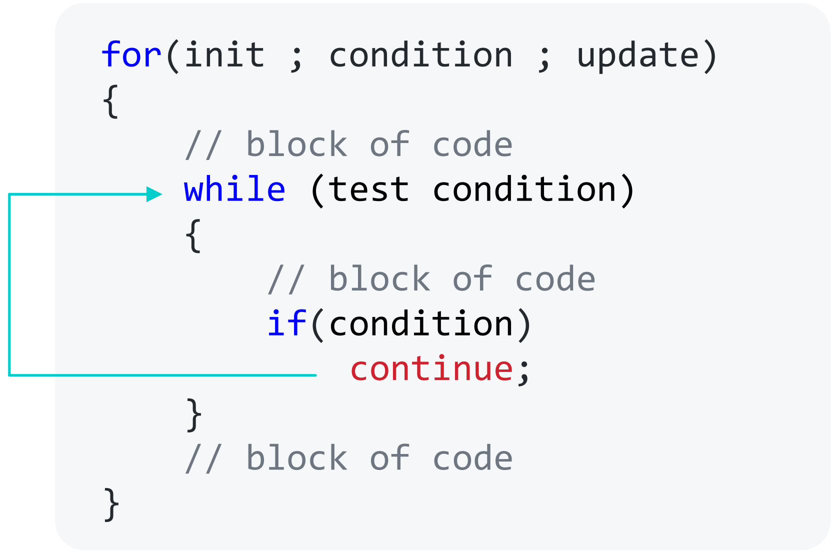
```cpp
#include <iostream>

using namespace std;

int main()
{
    for(int i=0 ; i<5 ; i++)
    {
        cout<<i<<": ";
        for(int j=1 ; j<=10 ; j++)
        {
            cout<<j<<" ";
        }
        cout<<endl;
    }

    return 0;
}
```

Nested Loops

- When we use a `break` statement inside the inner loop, it terminates the inner loop but not the outer loop.
- Similarly, when we use a `continue` statement inside the inner loop, it skips the current iteration of the inner loop only. The outer loop is unaffected.



```
for(init ; condition ; update)
{
    // block of code
    while (test condition)
    {
        // block of code
        if(condition)
            break;
    }
    // block of code
}
```



```
for(init ; condition ; update)
{
    // block of code
    while (test condition)
    {
        // block of code
        if(condition)
            continue;
    }
    // block of code
}
```

Example 1:

Given multiple lines each line contains a number x which is a password. Print "Wrong" if the password is incorrect otherwise, print "Correct" and terminate the program.

Note: The "Correct" password is the number 2021.

Input	Output
2200	Wrong
1020	Wrong
1000	Correct
2021	

** The answer of the example at github.*

Example 2:

Given two numbers X, Y. Print the sum of the numbers between X and Y.

Input	Output
2 20	209
1 100	5050
350 9500	45068675
1 1	1

** The answer of the example at github.*

Example 3:

Write a program to print the start pattern with N rows.

Input	Output
5	* * * * * * * * * * * * * * *

** The answer of the example at github.*

Example 4:

Write a program to print the start pattern with N rows.

Input	Output
5	* *

** The answer of the example at github.*

Example 5:

Write a program to print the hollow square star pattern with N rows and M columns.

Input	Output

5 5	`* * * * *`
	`*       *`
	`*       *`
	`*       *`
	`* * * * *`

** The answer of the example at github.*

References

Videos

- For Loop 1 (Mostafa Saad).
  - https://www.youtube.com/watch?v=LsBZvL4-KuE
- For Loop 2 (Mostafa Saad).
  - https://www.youtube.com/watch?v=_kjcb7w220c
- For Loop (Adel Nasim).
  - https://www.youtube.com/watch?v=cgpgpGuXIZk
- While Loop 1 (Mostafa Saad).
  - https://www.youtube.com/watch?v=qVBi98-XJ3s
- While Loop 2 (Mostafa Saad).
  - https://www.youtube.com/watch?v=yjzB3-CxWmE
- While Loop (Adel Nasim).
  - https://www.youtube.com/watch?v=YjiSuzc2pAM
- Nested Loop (Adel Nasim).
  - https://www.youtube.com/watch?v=uFI22C5DFnU
- Draw Shapes 1 (Adel Nasim).
  - https://www.youtube.com/watch?v=urzHVVJB17U
- Draw Shapes 2 (Adel Nasim).
  - https://www.youtube.com/watch?v=ow0-FArRzGE

THANKS!

Any Questions?!