Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Introduction Iteration (while/for)

Table of Contents

1. Introduction to Iteration 1

1.1 While loop 2

1.1.1 Components of *while* loop 4

1.1.2 An infinite loop 4

1.1.3 Loop control keywords 5

1.2 For loop 6

1.3 Nested loop 8

1. Introduction to Iteration

In programming, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several times.

Programming languages provide various control structures that allow for more complicated execution paths. A loop statement allows us to execute a statement or group of statements multiple times. Iteration and loop are the same things.

Programming language provides the following types of loops to handle looping requirements. It provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

While loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
For loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
Nested loops	It can use one or more loop inside any another while, for or do.

Example: If we need to print a block of code 5 times, instead of writing it multiple time we can use the loops-

```
Print ("Hello World")
Print ("Hello World")
Print ("Hello World")
Print ("Hello World")
Print ("Hello World")
```

Java(*while* loop):

```
int count=1;
while(count<=5){
    System.out.println("Hello World");
    Count = count + 1;
}
```

Python(*while* loop):

```
count=1
while count<=5:
    print("Hello World")
    Count = count + 1
```

Java(*for* loop):

```
for(int count=0; count<=5; count++){
    system.out.println( "Hello World");
}
```

Python(*for* loop):

```
for count in range(5):
    print("Hello World")
```

1.1 While loop

while loop is used to execute a block of statements repeatedly until a given condition is false, the line immediately after the loop in the program is executed.

- The syntax of a *while* loop in **Python** programming language is –

```
while expression:
    statement(s)
```

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. The loop iterates while the condition is true. When the condition becomes false, program control passes to the line immediately following the loop.

- The syntax of a *while* loop in **Java** programming language is –

```
while(expression) {
   // Statements
}
```

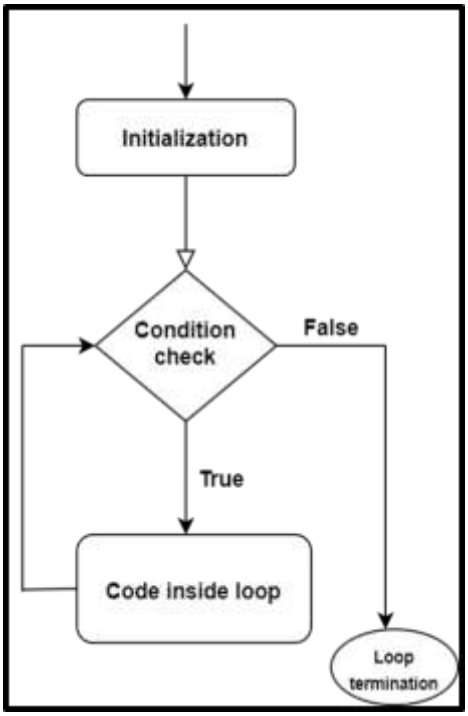
Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non-zero value. When executing, if the expression result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true. When the condition becomes false, program control passes to the line immediately following the loop.

Example: We have asked the program to execute the print command for 100 times by this.

```
count=1
while count<=100:
    print("hello world")
    count=count+1
```

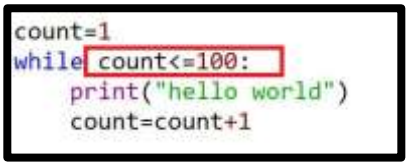
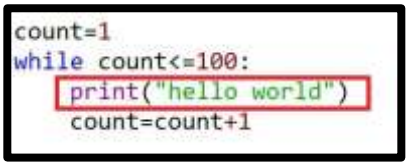
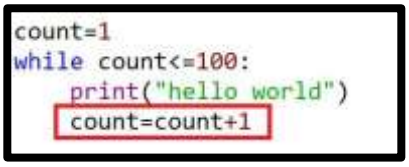
Here, count keeps track how many times program has executed the print command so far. It increases by one every time it executes the code. **while** count<=100: checks whether it has executed 100 times

Flow Diagram:



Here, key point of the *while* loop is that the loop might never run. When the expression is tested and the result is false, the loop body will be skipped and the first statement after the *while* loop will be executed.

1.1.1 Components of *while* loop

Initial value / initialization	The value from where the program will start counting.	
Condition	When the program should stop looping	
Block of code	The main code that should be executed several times	
Increment/decrement	Change of the loop control variable.	

1.1.2 An infinite loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

```
count=5
while count>=1:
    print(count)
    count=count+1
```

Here, the program is going to be executed for an infinite period of time because for every case the condition remains true.

1.1.3 Loop control keywords

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

- **Break:** It terminates the current loop and resumes execution at the statement after the loop.

  The most common use for *break* is when some external condition is triggered requiring a hasty exit from a loop. The *break* statement can be used in both while and for loops.

  If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block.

  The syntax for a break statement in **Python & Java** is as follows −

```
break
```

Example:

Here, when the value of count is 5, the condition breaks and execute the rest of the program.

Python

```
count = 1

while True:
  print(count,end=" " )
  count = count+1
  if count == 5:
   break
```

Java

```
int count = 1;

    while (count < 10) {
      System.out.print(count+" ");
      count++;
      if (count == 5) {
        break;
      }
 }
```

The output of following program for both java and python is **1 2 3 4**

- **Continue:** It returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both *while* and *for* loops.

The syntax for a break statement in **Python & Java** is as follows –

```
continue
```

Example:

Here, when the value of count is 3, the program is going to skip the lines inside the loop and start again from condition check.

Python

```
count = 0

while count<5:
  count = count+1
  if count == 3:
   continue
  print(count,end=" ")
```

Java

```
int count = 0;
    while (count < 5) {
      count++;
      if (count == 3) {
        continue;
      }
      System.out.print(count+" ");
    }
```

The output of following program for both java and python is **1 2 4 5**

1.2 For loop

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. *for* statement iterates over the members of a sequence in order, executing the block each time. Contrast the *for* statement with the *while* loop, used when a condition needs to be checked each iteration, or to repeat a block of code forever.

- The syntax of a *for* loop in **Python** programming language is –

```
for iterating_var in sequence:
   statement(s)
```

Here, if a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable **iterating_var.** Next, the statements block is executed. Each item in the list is assigned to **iterating_var** and the **statement(s)** block is executed until the entire sequence is exhausted.

- The syntax of a *for* loop in **Java** programming language is −

```
for(initialization; Boolean_expression; update) {
   // Statements
}
```

Here is the flow of control in a **for** loop −

- The **initialization** step is executed first and only once. This step allows you to declare and initialize any loop control variables and this step ends with a semi colon (;).
- Next, the **Boolean expression** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the *for* loop.
- After the **body** of the *for* loop gets executed, the control jumps back up to the update statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
- The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the *for* loop terminates.

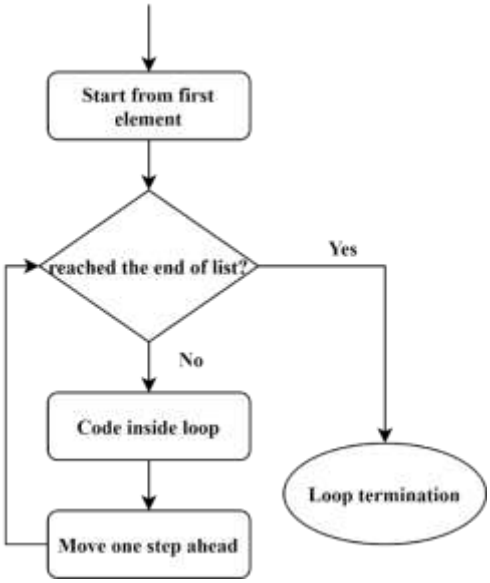
Example: Here we have asked the program to execute the print command for the given list.

```
a=[1,2,3,4,5]

for i in a:
    print(i)
```

Here, the variable **i** starts from the first element of the list. In each iteration **i** moves forward. Then the loop ends when **i** reaches the end of a list.

Flow diagram:



1.3 Nested loop

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept. A final note on loop nesting is that you can put any type of loop inside of any other type of loop.

Syntax:

```
for iterating_var in sequence:
   for iterating_var in sequence:
      statement(s)
   statement(s)
```

- The syntax for a nested while loop statement in Python programming language is as follows −

```
while expression:
   while expression:
      statement(s)
   statement(s)
```

Example:

```
outer=1
while outer <= 2:
  inner=1
  while inner <= 3:
    print(outer, "and", inner)
    inner = inner + 1
  print("inner loop terminates")

  outer = outer + 1
print("outer loop terminates")
```

Output

1 and 1

1 and 2

1 and 3

inner loop terminates

2 and 1

2 and 2

2 and 3

inner loop terminates

outer loop terminates

Flow diagram: