6.1 Program Flow and Control

A program's control flow is the order in which the program's code executes. The control flow of a Python program is regulated by conditional statements, loops, and function calls. This section covers the if statement and for and while loops; functions are covered later in this chapter.

Python has three types of control structures:

- Sequential - default mode
- Selection - used for decisions and branching
- Repetition - used for looping, i.e., repeating a piece of code multiple times.

1. Sequential
Sequential statements are a set of statements whose execution process happens in a sequence. The problem with sequential statements is that if the logic has broken in any one of the lines, then the complete source code execution will break.

```
1 a=20
2 b=10
3 c=a-b
4 print("Subtraction is : ",c)
```

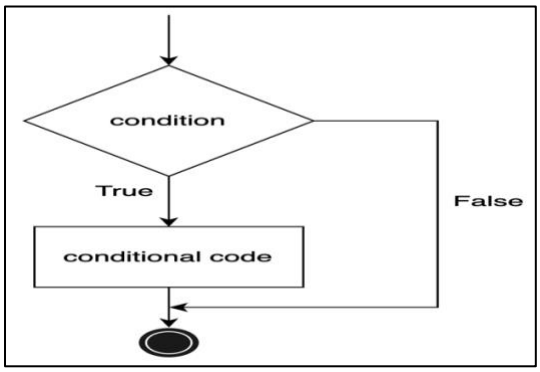
```
muthanna@maxo79:~/Desktop$ python3 sequential.py
Subtraction is :  10
```

2. Selection/Decision control statements
In Python, the selection statements are also known as Decision control statements or branching statements.
The selection statement allows a program to test several conditions and execute instructions based on which condition is true.
Some decision control statements are:

```
if expression:
    statement(s)
elif expression:
    statement(s)
elif expression:
    statement(s)
...
else:
    statement(s)
```

- **if**

It help us to run a particular code, but only when a certain condition is met or satisfied. A if only has one condition to check.



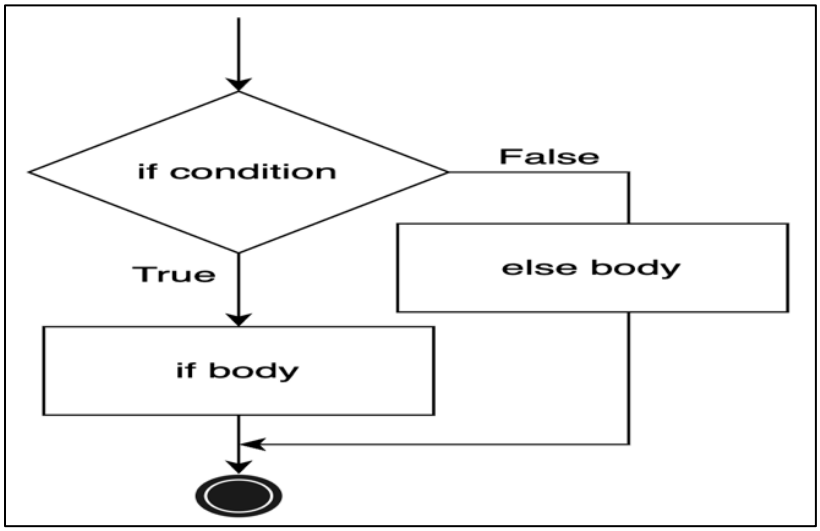
In the following code example, we're checking the if condition, e.g., if n modulo 2 equals to 0 then the print statement will execute.

```
1 n=10
2 if n%2==0:
3     print ("even")
```

```
muthanna@maxo79:~/Desktop$ python3 if.py
even
```

- **if-else**

The if-else statement evaluates the condition and will execute the body of if if the test condition is True, but if the condition is False, then the body of else is executed.



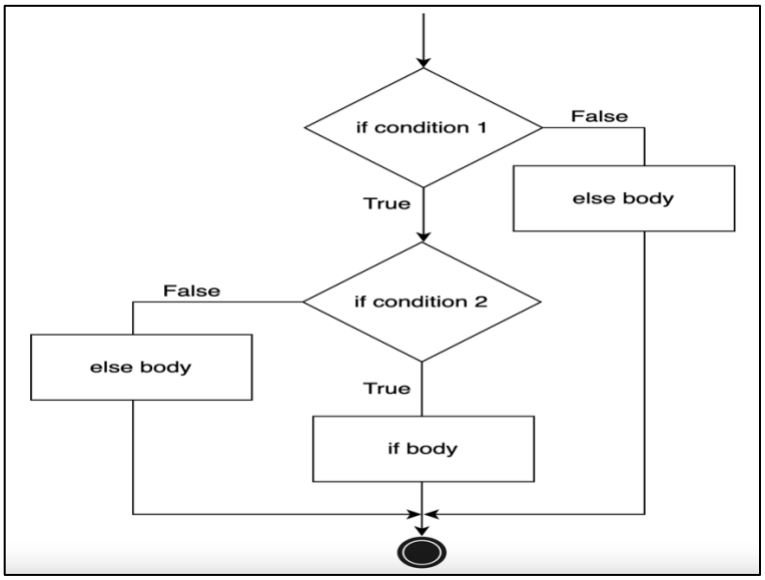
In the following code example, the else body will execute as 5 modulo 2 is not equal to 0.

```
n=7
if n%2==0:
    print ("even")
else:
    print ("odd")
```

```
muthanna@maxo79:~/Desktop$ python3 if.py
odd
```

Nested if

Nested if statements are an if statement inside another if statement.



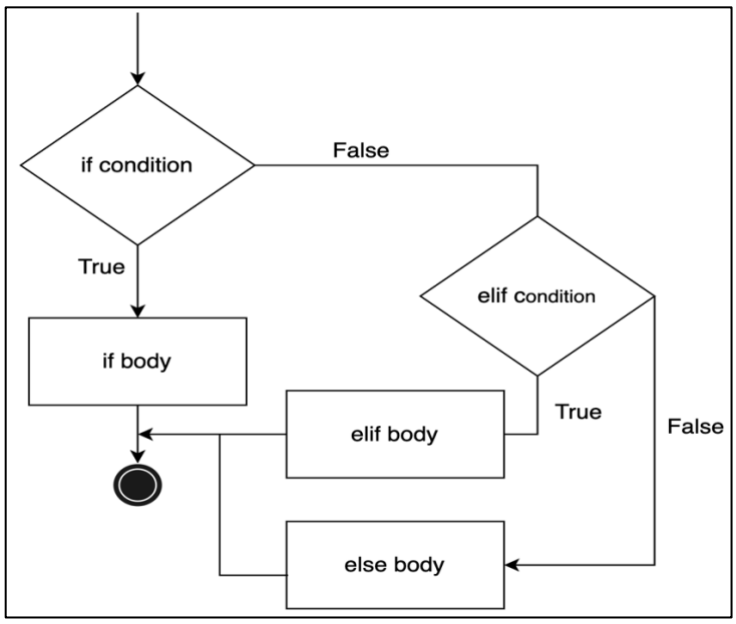
In the following code example, we can see first if condition checks a is greater than b. If yes, then we've another if condition that checks a is also greater than c. If yes, then if body will be executed.

```
a = 20
b = 10
c = 15
if a > b:
   if a > c:
      print("a value is big")
   else:
       print("c value is big")
elif b > c:
    print("b value is big")
else:
     print("c is big")
```

```
muthanna@maxo79:~/Desktop$ python3 if.py
a value is big
```

- **if-elif-else**

The if-elif-else statement is used to conditionally execute a statement or a block of statements.



In the code provided, the initial if statement checks whether 15 is equal to 12. If not, the elif condition will verify that 15 is greater than 12. If this condition is true, the code block inside elif will be executed. Otherwise, the code in the else block will be executed if none of the preceding conditions are met.

```
1 x = 15
2 y = 12
3 if x == y:
4     print("Both are Equal")
5 elif x > y:
6     print("x is greater than y")
7 else:
8     print("x is smaller than y")
```

```
muthanna@maxo79:~/Desktop$ python3 if.py
x is greater than y
```

3. Repetition

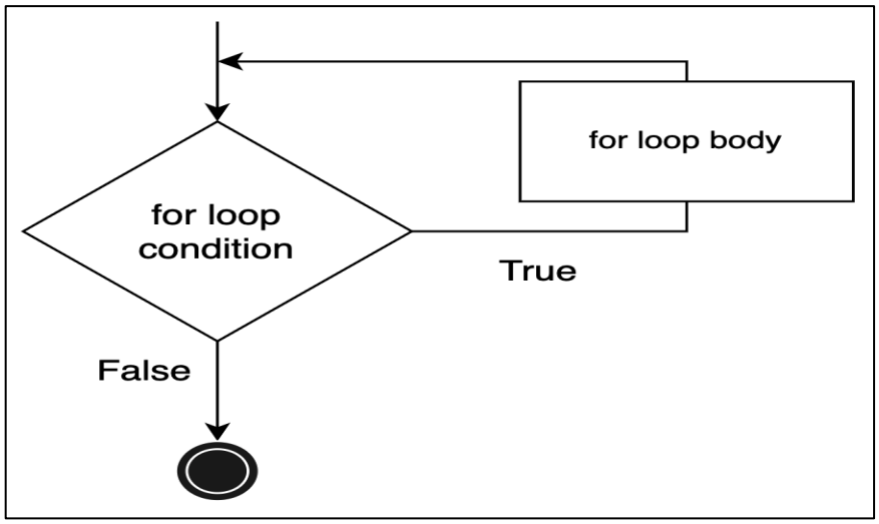
A repetition statement is used to repeat a group(block) of programming instructions.

In Python, we generally have two loops/repetitive statements:

- for loop
- while loop

- **for loop**

A for loop is used to iterate over a sequence that is either a list, tuple, dictionary, or a set. We can execute a set of statements once for each item in a list, tuple, or dictionary.



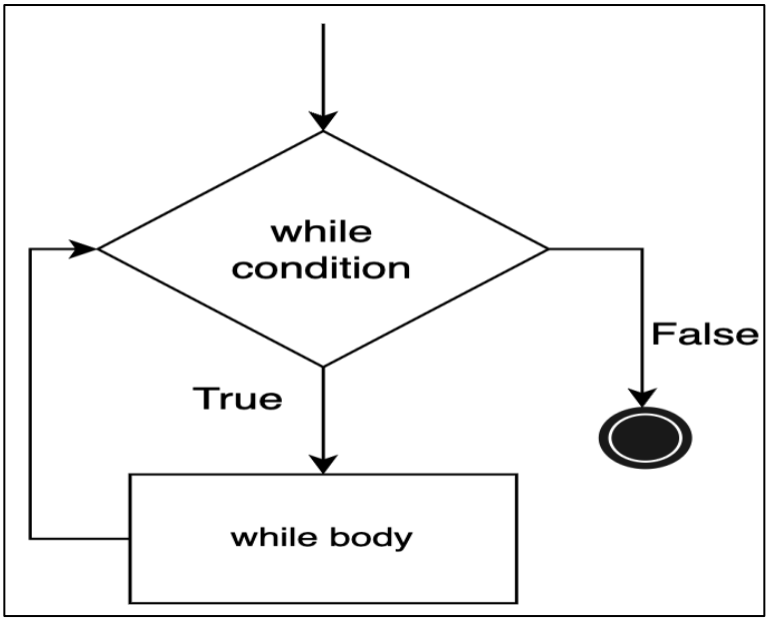
In the following code example, the first example demonstrates how to use a for loop to iterate over a list and access its elements, while the second example shows how to iterate over a range of numbers.

```
1 print("1st example")
2 lst = [1, 2, 3]
3 for i in range(len(lst)):
4     print(lst[i])
5
6 print("2nd example")
7 for j in range(0,5):
8     print(j)
```

```
muthanna@maxo79:~/Desktop$ python3 for.py
1st example
1
2
3
2nd example
0
1
2
3
4
```

- **while loop**

In Python, while loops are used to execute a block of statements repeatedly until a given condition is satisfied. Then, the expression is checked again and, if it is still true, the body is executed again. This continues until the expression becomes false.



The following code iterates from 0 to 4 and prints each value using a while loop. It prints End to signify the end of the program after the loop is completed.

```
1 m = 5
2 i = 0
3 while i < m:
4       print(i, end = " ")
5       i = i + 1
6 print("End")
```

```
muthanna@maxo79:~/Desktop$ python3 while.py
0 1 2 3 4 End
```

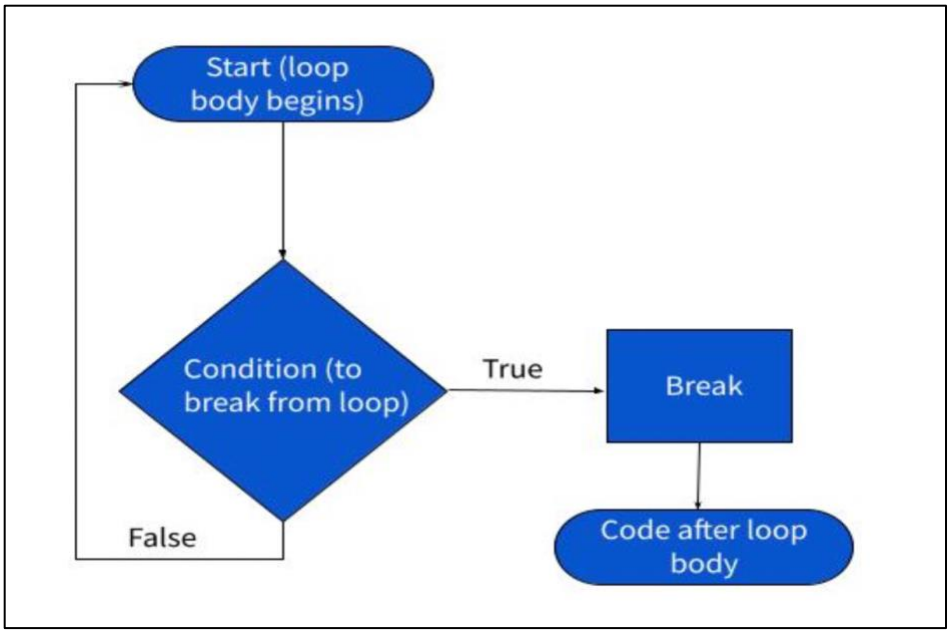
Homework: what is output in previously example if we delete (end=" ") or ("End ") or booth in print function? and why?

6.2 Break and Continue

Break and continue are two ways to modify the behavior of for loops and while loops.

- **Break**

In Python, the keyword break causes the program to exit a loop early. break causes the program to jump out of for loops even if the for loop hasn't run the specified number of times.break causes the program to jump out of while loops even if the logical condition that defines the loop is still True.



An example using break in a for loop is below.

```
1 for i in range(100):
2     print(i)
3     if i == 3:
4         break
5 print('Loop exited')
```

```
muthanna@maxo79:~/Desktop$ python3 break.py
0
1
2
3
Loop exited
```

When the loop hits i=3, break is encountered and the program exits the loop.
An example using break in a while loop is below.

```
m = 0
while m < 11:
      print(m)
      m = m + 1
      if m==6:
          break
```

```
muthanna@maxo79:~/Desktop$ python3 while.py
0
1
2
3
4
5
```

- **Continue**

In Python, the keyword continue causes the program to stop running code in a loop and start back at the top of the loop. Remember the keyword break cause the program to exit a loop. continue is similar, but continue causes the program to stop the current iteration of the loop and start the next iteration at the top of the loop.

A code section that uses continue in a for loop is below.

```
for i in range(6):
    if i == 4:
        continue
    print(i)
```

```
muthanna@maxo79:~/Desktop$ python3 break.py
0
1
2
3
5
```

A code section that uses continue in a while loop is below.

```
num = 10
while num < 100:
    num = num + 10
    if num==50:
        continue
    print(num)
```

```
muthanna@maxo79:~/Desktop$ python3 while.py
20
30
40
60
70
80
90
100
```