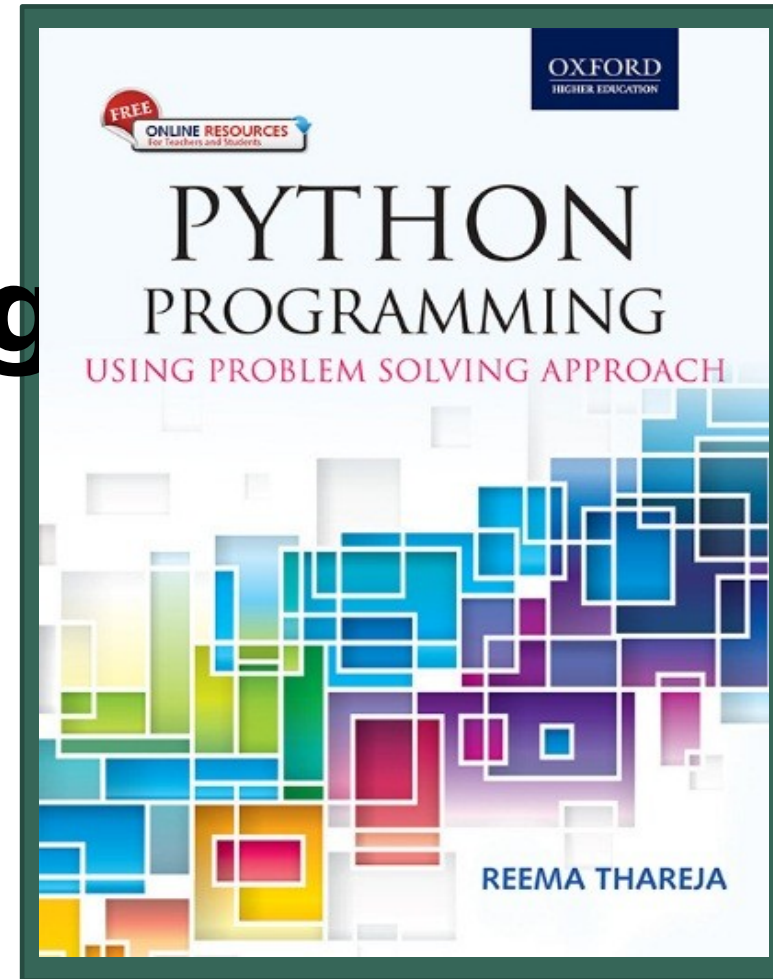




Python Programming

Using Problem Solving Approach

Reema Thareja



1



CHAPTER 4

Decision Control Statements

Control Statements

A **control statement** is a statement that determines the control flow of a set of instructions, i.e., it decides the sequence in which the instructions in a program are to be executed.

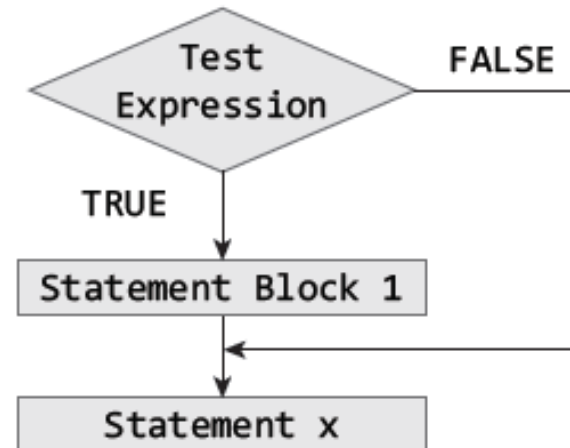
Types of Control Statements —

- **Sequential Control:** A Python program is executed sequentially from the first line of the program to its last line.
- **Selection Control:** To execute only a selected set of statements.
- **Iterative Control:** To execute a set of statements repeatedly.

If Statement

SYNTAX OF IF STATEMENT

```
if test_expression:  
    statement1  
    .....  
    Statement n  
statement x;
```



Example:

```
x = 10      #Initialize the value of x  
if(x>0):    #test the value of x  
    x = x+1  #Increment the value of x if it is > 0  
print(x)    #Print the value of x
```

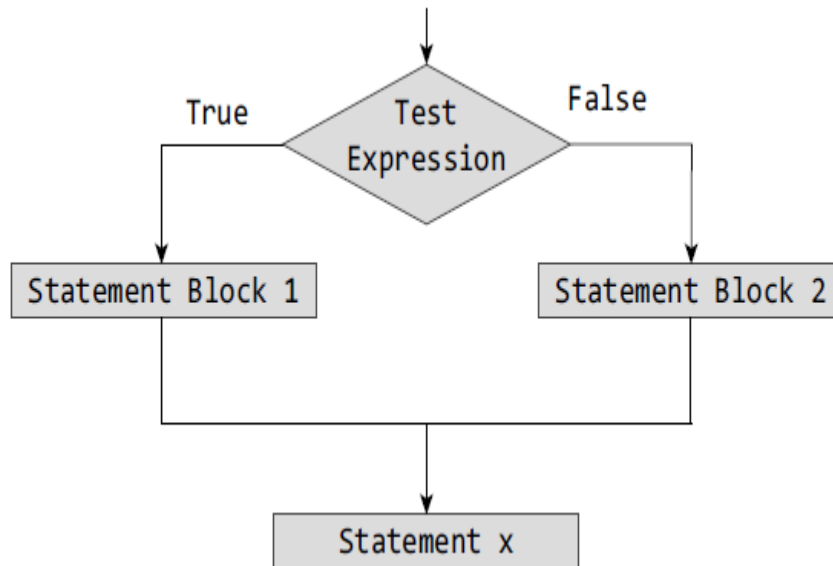
OUTPUT

```
x = 11
```

If-Else Statement

SYNTAX OF IF-ELSE STATEMENT

```
if (test expression):  
    statement block 1  
else:  
    statement block 2  
statement x;
```



Example:

```
age = int(input("Enter the age : "))  
if(age>=18):  
    print("You are eligible to vote")  
else:  
    yrs = 18 - age  
    print("You have to wait for another " + str(yrs) + " years to cast your vote")
```

OUTPUT

```
Enter the age : 10  
You have to wait for another 8 years to cast your vote
```

Nested if Statements

A statement that contains other statements is called a ***compound statement***. To perform more complex checks, if statements can be nested, that is, can be placed one inside the other. In such a case, the inner if statement is the statement part of the outer one. Nested if statements are used to check if more than one conditions are **satisfied**.

Example:

```
num = int(input("Enter any number from 0-30: "))
if(num>=0 and num<10):
    print("It is in the range 0-10")
elif(num>=10 and num<20):
    print("It is in the range 10-20")
elif(num>=20 and num<30):
    print("It is in the range 20-30")
```

OUTPUT

```
Enter any number from 0-30: 25
It is in the range 20-30
```

If-elif-else Statement

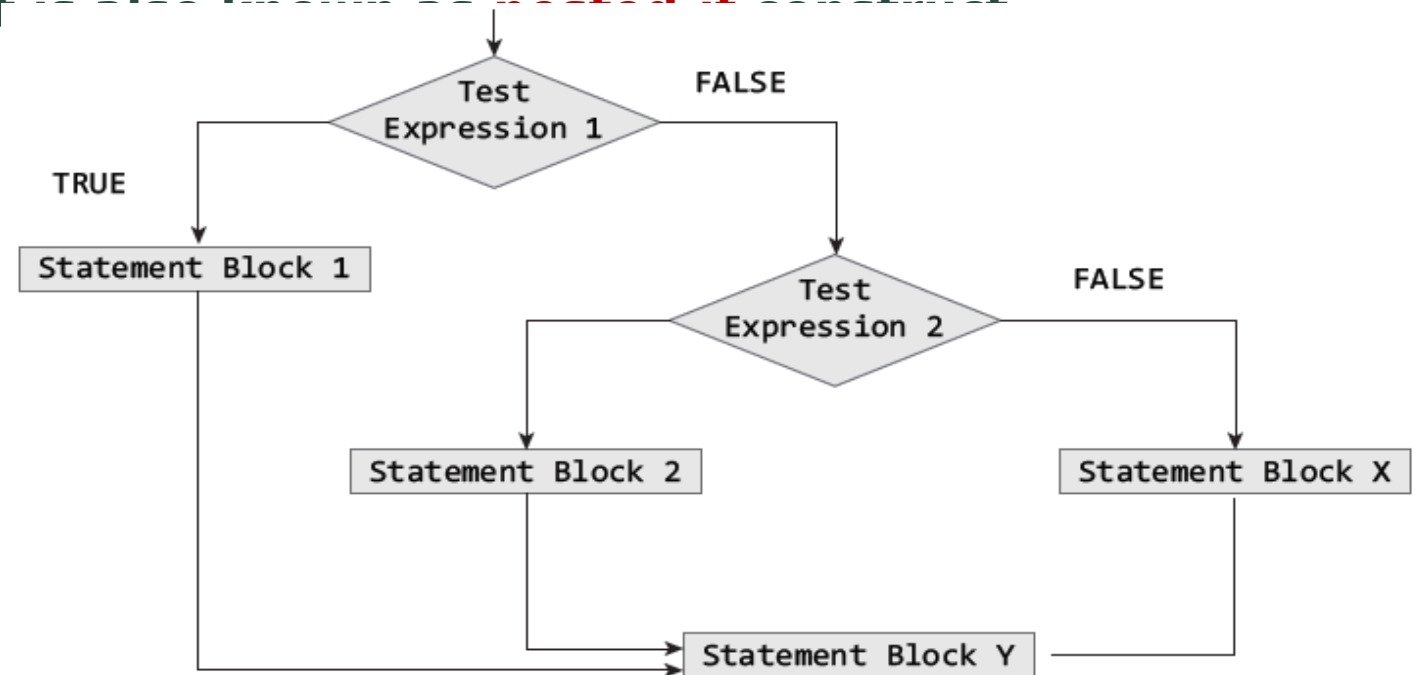
Python supports if-elif-else statements to test additional conditions apart from the initial test expression. The if-elif-else construct works in the same way as a usual if-else statement. If-elif-else construct

Example:

```
num = int(input("Enter any number : "))
if(num==0):
    print("The value is equal to zero")
elif(num>0):
    print("The number is positive")
else:
    print("The number is negative")
```

OUTPUT

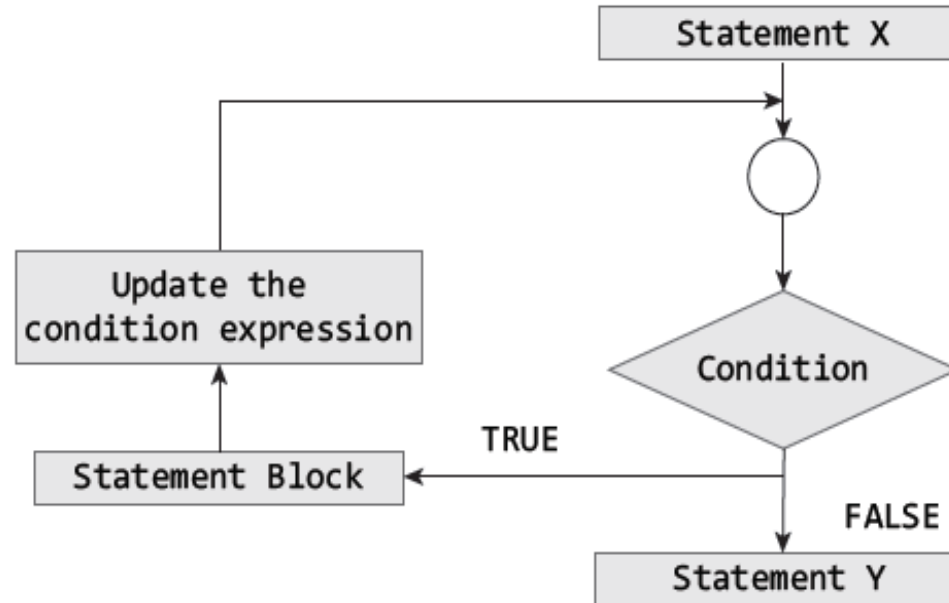
```
Enter any number : -10
The number is negative
```



While Loop

SYNTAX OF WHILE LOOP

```
statement x  
while (condition):  
    statement block  
statement y
```



Example:

```
i = 0  
while(i<=10):  
    print(i,end=" ")  
    i = i+1
```

OUTPUT

0 1 2 3 4 5 6 7 8 9 10

For Loop

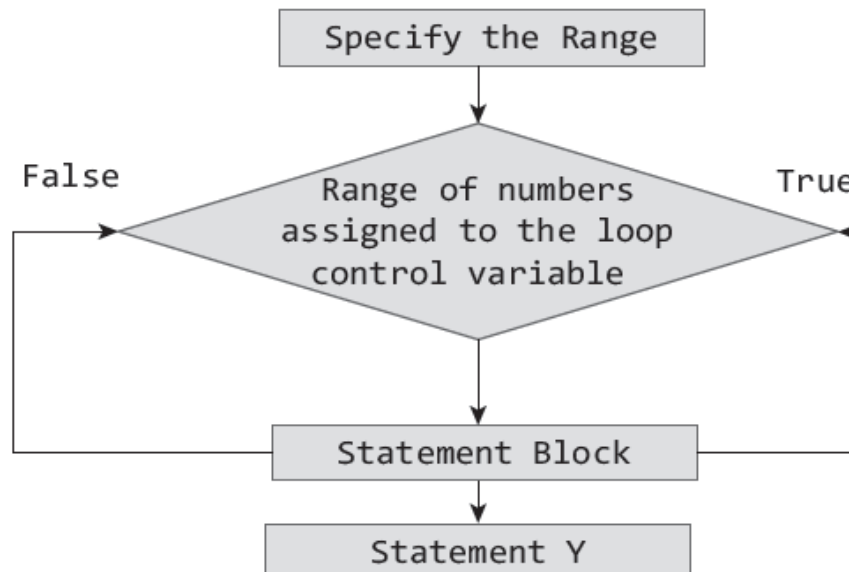
For loop provides a mechanism to repeat a task until a particular condition is True. It is usually known as a *determinate or definite loop* because the programmer knows exactly how many times the loop will repeat.

The `for...in` statement is a looping statement used in Python to iterate over a sequence of

Syntax of for Loop

```
for loop_control_var in sequence:  
    statement block
```

Flow of Statements in for loop



For Loop and Range() Function

The **range()** function is a built-in function in Python that is used to iterate over a sequence of numbers. The syntax of range() is **range(beg, end, [step])**

The range() produces a sequence of numbers starting with beg (inclusive) and ending with one less than the number end. The step argument is option (that is why it is placed in brackets). By default, every number in the range is incremented by 1 but we can specify a different increment using step. It can be both negative and positive, but not zero.

Examples:

```
for i in range(1, 5):  
    print(i, end= " ")
```

OUTPUT

1 2 3 4

Print numbers
in the same line

```
for i in range(1, 10, 2):  
    print(i, end= " ")
```

OUTPUT

1 3 5 7 9

beg step
end

Range() Function

If `range()` function is given a single argument, it produces an object with values from 0 to argument-1. For example: `range(10)` is equal to writing `range(0, 10)`.

- If `range()` is called with two arguments, it produces values from the first to the second. For example, `range(0,10)`.

- If `range()` has three arguments then the third argument specifies the interval of the sequence produced. In this case, the third argument must be an integer. For example, `range(1,20,3)`.

Examples:

```
for i in range(10):  
    print (i, end= ' ')
```

OUTPUT

0 1 2 3 4 5 6 7 8 9

```
for i in range(1,15):  
    print (i, end= ' ')
```

OUTPUT

1 2 3 4 5 6 7 8 9 10 11 12 13 14

```
for i in range(1,20,3):  
    print (i, end= ' ')
```

OUTPUT

1 4 7 10 13 16 19

Condition-controlled and Counter-controlled Loops

Attitude	Counter-controlled loop	Condition controlled loop
Number of execution	Used when number of times the loop has to be executed is known in advance.	Used when number of times the loop has to be executed is not known in advance.
Condition variable	In counter-controlled loops, we have a counter variable.	In condition-controlled loops, we use a sentinel variable.
Value and limitation of variable	The value of the counter variable and the condition for loop execution, both are strict.	The value of the counter variable and the condition for loop execution, both are strict.
Example	<pre>i = 0 while(i<=10): print(i, end = " ") i+=1</pre>	<pre>i = 1 while(i>0): print(i, end = " ") i+=1 if(i==10): break</pre>

Nested Loops

Python allows its users to have nested loops, that is, loops that can be placed inside other loops. Although this feature will work with any loop like while loop as well as for loop.

A for loop can be used to control the number of times a particular set of statements will be executed. Another outer loop could be used to control the number of times that a whole loop is repeated.

Example:
Loops should
each for sta

```
*****  
*****  
*****  
*****  
*****  
  
for i in range(5):  
    print()  
    for j in range(5):  
        print("*",end=' ')
```

identify which statements are contained within

The Break Statement

The *break* statement is used to terminate the execution of the nearest enclosing loop in which it appears. The break statement is widely used with for loop and while loop. When compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears.

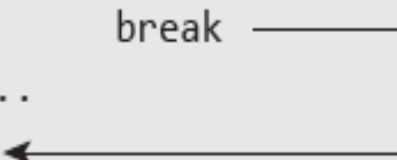
Example:

```
i = 1
while i <= 10:
    print(i, end=" ")
    if i==5:
        break
    i = i+1
print("\n Done")
```

OUTPUT

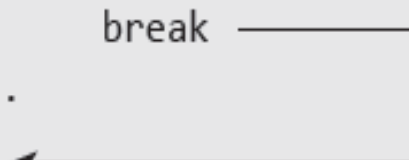
```
1 2 3 4 5
Done
```

```
while...
.....
if condition:
    break
.....
.....
```



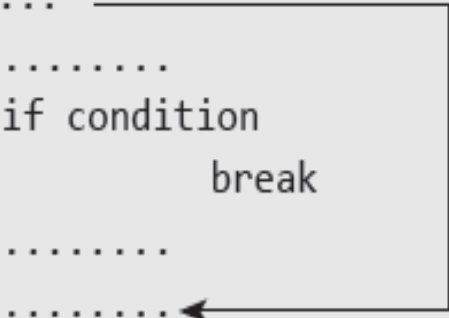
Transfers control out of the while loop

```
for...
.....
if condition:
    break
.....
.....
```



Transfers control out of the for loop

```
for...
.....
for...
.....
if condition:
    break
.....
.....
```



Transfers control out of inner for loop

The Continue Statement

Like the break statement, the continue statement can only appear in the body of a loop. When the compiler encounters a continue statement then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.

Example:

```
for i in range(1,11):
    if(i==5):
        continue
    print(i, end=" ")
print("\n Done")
```

OUTPUT

```
1 2 3 4 6 7 8 9 10
Done
```

```
while(...) ←
...
    If condition:
        continue
...
Transfers control to the condition
expression of the while loop
```

```
for(...) ←
...
    if condition:
        continue
...
Transfers control to the condition
expression of the for loop
```

```
for(...)
...
    for(...) ←
        ...
        if condition:
            continue
        ...
    ...
Transfers control to the condition
expression of the inner for loop
```

The Pass Statement

Pass statement is used when a statement is required syntactically but no command or code has to be executed. It specifies a *null* operation or simply No Operation (NOP) statement. Nothing happens when the pass statement is executed.

Difference between comment and pass statements In Python programming, pass is a null statement. The difference between a comment and pass statement is that while the interpreter ignores a comment entirely, pass is not ignored. Comment is not executed

but pass statement is executed.
Example:

```
for letter in "HELLO":  
    pass      #The statement is doing nothing  
    print("Pass : ", letter)  
print("Done")
```

OUTPUT

```
Pass :  H  
Pass :  E  
Pass :  L  
Pass :  L  
Pass :  O  
Done
```


The Else Statement Used With Loops

Unlike C and C++, in Python you can have the *else* statement associated with a loop statements. If the *else* statement is used with a *for* loop, the *else* statement is executed when the loop has completed iterating. But when used with the *while* loop, the *else* statement is executed when the condition becomes false.

Examples:

```
for letter in "HELLO":
    print(letter, end=" ")
else:
    print("Done")
```

OUTPUT

H E L L O Done

```
i = 1
while(i<0):
    print(i)
    i = i - 1
else:
    print(i, "is not negative so
loop did not execute")
```

OUTPUT

1 is not negative so loop did not execute