Chapter 3

Conditional and Looping Construct

After studying this lesson, students will be able to:

- *Understand the concept and usage of selection and iteration statements.*
- *Know various types of loops available in Python.*
- *Analyze the problem, decide and evaluate conditions.*
- *Will be able to analyze and decide for an appropriate combination of constructs.*
- *Write code that employ decision structures, including those that employ sequences of decision and nested decisions.*
- *Design simple applications having iterative nature.*

Control Flow Structure

Such as depending on time of the day you wish Good Morning or Good Night to people. Similarly while writing program(s), we almost always need the ability to check the condition and then change the course of program, the simplest way to do so is using **if** statement

```
if x > 0:
   print 'x is positive'
```

Here, the Boolean expression written after **if** is known as condition, and if Condition is **True,** then the statement written after, is executed. Let's see the syntax of **if** statement

Option 1	Option 2

if condition:
   STATEMENTs- **BLOCK 1**
[else:
	if condition-1:
	   STATEMENTs- **BLOCK 1**
	[elif condition-2:

STATEMENTs- **BLOCK 2**]	STATEMENTs- **BLOCK 2** else: STATEMENTs- **BLOCK N**]

Statement with in [ ] bracket are optional.

Let us understand the syntax, in Option 1- if the condition is **True** (i.e. satisfied), the statement(s) written after **if** (i.e. STATEMENT-BLOCK 1) is executed, otherwise statement(s) written after else (i.e. STATEMENT-BLOCK 2) is executed. Remember **else** clause is optional. If provided, in any situation, one of the two blocks get executed not both.

We can say that, 'if' with 'else' provides an alternative execution, as there are two possibilities and the condition determines which one gets executed. If there are more than two possibilities, such as based on percentage print grade of the student.

Percentage Range	**Grade**
> 85	A
> 70 to <=85	B
> 60 to <=70	C
> 45 to <=60	D

Then we need to chain the **if** statement(s). This is done using the 2nd option of **if** statement. Here, we have used **'elif'** clause instead of 'else'. **elif** combines **if else- if else** statements to one **if elif …else**. You may consider elif to be an abbreviation of *else if*. There is no limit to the number of 'elif' clause used, but if there is an 'else' clause also it has to be at the end.

Example for combining more than one condition:

```
if perc > 85:
    print 'A'
elif perc >70 and perc <=85:          #alternative to this is if 70 <perc<85
```

```
        print 'B'
    elif perc > 60 and perc <=70:           #if 60 <perc <=70
        print 'C'
    elif perc >45 and perc <=60:
        print 'D'
```

In the chained conditions, each condition is checked in order– if previous is F**alse** then next is checked, and so on. If one of them is **True** then corresponding block of statement(s) are executed and the statement ends i.e., control moves out of 'if statement'. If none is true, then else block gets executed if provided. *If more than one condition is true, then only the first true option block gets executed.*

If you look at the conditional construct, you will find that it has same structure as function definition, terminated by a colon. Statements like this are called compound statements. In any compound statement, there is no limit on how many statements can appear inside the body, but there has to be at least one. Indentation level is used to tell Python which statement (s) belongs to which block.

There is another way of writing a simple if else statement in Python. The complete simple if, can be written as:

Variable= variable 1 if condition else variable 2.

In above statement, on evaluation, if condition results into True then variable 1 is assigned to Variable otherwise variable 2 is assigned to Variable.

Example

```
>>> a =5
>>> b=10
>>> x = True
>>> y = False
>>>result = x if a <b else y
```

Will assign **True** to result

Sometimes, it is useful to have a body with no statements, in that case you can use **pass** statement. Pass statement does nothing.

Example

```
if condition:
pass
```

It is possible to have a condition within another condition. Such conditions are known as **Nested Condition.**

Example

```
if x==y:
    print x, ' and ', y, ' are equal'
else:
if x<y:
    print x, ' is less than ', y          } Nested if
else:
    print x, ' is greater than ', y
```

Here a complete **if… else** statement belongs to else part of outer if statement.

Note: The condition can be any Python expression (i.e. something that returns a value). Following values, when returned through expression are considered to be False:

None, Number Zero, A string of length zero, an empty collection

Looping Constructs

We know that computers are often used to automate the repetitive tasks. One of the advantages of using computer to repeatedly perform an identical task is that –it is done without making any mistake. Loops are used to repeatedly execute the same code in a program. Python provides two types of looping constructs:

1) While statement

2) For statement

While Statements

Its syntax is:

```
while condition:                # condition is Boolean expression returning True or False
   STATEMENTs BLOCK 1
[else:                          # optional part of while
   STATEMENTs BLOCK 2]
```

We can see that while looks like if statement. The statement bExampleins with keyword **while** followed by boolean condition followed by colon (:). What follows next is block of statement(s).

The statement(s) in BLOCK 1 keeps on executing till condition in while remains **True;** once the condition becomes **False** and if the else clause is written in while, then else will get executed. While loop may not execute even once, if the condition evaluates to false, initially, as the condition is tested before entering the loop.

Example

a loop to print nos. from 1 to 10

```
i=1
while (i <=10):
    print i,
    i = i+1        #could be written as i+=1
```

You can almost read the statement like English sentence. The first statement initialized the variable (controlling loop) and then **while** evaluates the condition, which is True so the block of statements written next will be executed.

Last statement in the block ensures that, with every execution of loop, **loop control variable** moves near to the termination point. If this does not happen then the loop will keep on executing infinitely.

As soon as **i becomes 11**, condition in while will evaluate to **False** and this will terminate the loop. Result produced by the loop will be:

1 2 3 4 5 6 7 8 9 10

As there is ',' after print i all the values will be printed in the same line

Example

```
i=1
while (i <=10):
    print i,
    i+ =1
else:
    print              # will bring print control to next printing line
    print "coming out of loop"
```

Will result into

1 2 3 4 5 6 7 8 9 10

coming out of loop

Nested loops

Block of statement belonging to while can have another while statement, i.e. a while can contain another while.

Example

```
i=1
while i<=3:
    j=1
    while j<=i:  }
    print j,     } # inner while loop
    j=j+1
```

```
        print
        i=i+1
```

will result into

```
1
1 2
1 2 3
```

For Statement

Its Syntax is

```
for TARGET- LIST in EXPRESSION-LIST:
   STATEMENT BLOCK 1
[else:                          # optional block
   STATEMENT BLOCK 2]
```

Example

```
# loop to print value 1 to 10
for i in range (1, 11, 1):
    print i,
```

Execution of the loop will result into

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

Let's understand the flow of execution of the statement:

The statement introduces a function range ( ), its syntax is

```
range(start, stop, [step])          # step is optional
```

range( ) generates a list of values starting from **start** till **stop-1**. Step if given is added to the value generated, to get next value in the list. *You have already learnt about it in built-in functions.*

Let's move back to the for statement: **i** is the variable, which keeps on getting a value generated by range ( ) function, and the block of statement (s) are worked on for each value of **i**. As the last value is assigned to **i**, the loop block is executed last time and control is returned to next statement. If **else** is specified in **for** statement, then next statement executed will be else. Now we can easily understand the result of **for** statement. range( ) generates a list from 1, 2, 3, 4, 5, …., 10 as the step mentioned is 1, **i** keeps on getting a value at a time, which is then printed on screen.

Apart from range( ) i (loop control variable) can take values from string, list, dictionary, etc.

Example

```
for letter in 'Python':
    print 'Current Letter', letter
else:
    print 'Coming out of loop'
```

On execution, will produce the following:

```
Current Letter: P
Current Letter: y
Current Letter: t
Current Letter: h
Current Letter: o
Current Letter: n
Coming out of loop
```

A **for** statement can contain another **for** statement or **while** statement. We know such statement form nested loop.

Example

```
# to print table starting from 1 to specified no.
n=2
for i in range (1, n+1):
    j=1
```

```
        print "Table to ", i, "is as follows"
        while j <6:
    print i, "*", j "=", i*j
        j = j+1
        print
```

Will produce the result

```
Table to 1 is as follows
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
Table to  2 is as follows
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
```

Nesting a **for loop** within **while loop** can be seen in following example :

Example

```
i = 6
while i >= 0:
    for j in range (1, i):
        print j,
    print
```

```
          i=i-1
```

will result into

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

By now, you must have realized that, Syntax of **for statement** is also same as **if statement** or **while statement.**

Let's look at the equivalence of the two looping construct:

While	**For**

i= initial value
while (i <limit):
    statement(s)
    i+=step
	for i in range (initial value, limit, step):
	    statement(s)

Break Statement

Break can be used to unconditionally jump out of the loop. It terminates the execution of the loop. Break can be used in while loop and for loop. Break is mostly required, when because of some external condition, we need to exit from a loop.

Example

```
for letter in 'Python':
if letter = = 'h':
     break
     print letter
```

will result into

```
P

y

t
```

Continue Statement

This statement is used to tell Python to skip the rest of the statements of the current loop block and to move to next iteration, of the loop. Continue will return back the control to the bExampleinning of the loop. This can also be used with both while and for statement.

Example

```
for letter in ‘Python’:
if letter == ‘h’:
    continue
    print letter
will result into
```

```
P
y
t
o
n
```

EXERCISE

1) Mark True/False:

   (i) While statements gets executed at least once

   (ii) The break statement allows us to come out of a loop

   (iii) The continue and break statement have same effect

   (iv) We can nest loops

   (v) We cannot write a loop that can execute forever.

   (vi) Checking condition in python can be done by using the if-else statement

2) What is the difference between the following two statements:

   (i)
```
 if n>2:
if n <6 :
        print 'OK'
    else:
        print 'NG'
```

   (ii)
```
if n>2:
    if n<6:
        print 'OK'
    else:
        print 'NG'
```

3) Mark the correct Option(s)

   (i) If there are two or more options, then we can use

a)	Simple if statement	b)	If elif statement
c)	While	d)	None of these

(ii) A loop that never ends is called a:

a) Continue loop   b) Infinite loop

c) Circle loop   d) None of these

4) Construct a logical expression to represent each of the following conditions:

(i) Score is greater than or equal to 80 but less than 90

(ii) Answer is either 'N' or 'n'

(iii) N is between 0 and 7 but not equal to 3

5) Which of the following loop will continue infinitely:

(i) a) while O: b) while 1:

c) while :1:   d) while False:

(ii) We can go back to the start of the loop by using ___________

a) loop   b) back

c) start   d) continue

6) What is the difference between selection and repetition?

7) Explain use of if statement with example.

LAB EXERCISE

1)
```
answer = raw_input("Do you like Python? ")
if answer == "yes":
print "That is great!"
else:
    print "That is disappointing!"
```

Modify the program so that it answers "That is great!" if the answer was "yes", "That is disappointing" if the answer was "no" and "That is not an answer to my question." otherwise.

2) Write a function to find whether given number is odd or even.

3) Print all multiples of 13 that are smaller than 100. Use the range function in the following manner: range (start, end, step) where "start" is the starting value of the counter, "end" is the end value and "step" is the amount by which the counter is increased each time.

4) Write a program using while loop that asks the user for a number, and prints a countdown from that number to zero. **Note:** Decide on what your program will do, if user enters a nExampleative number.

5) Using for loop, write program that prints out the decimal equivalent of ½, $^1/_3$, ¼, ------, $^1/_{10}$

6) Write a function to print the Fibonacci Series up to an Input Limit.

7) Write a function to generate and print factorial numbers up to *n* (provided by user).

8) Write a program using a for loop, that calculates exponentials. Your program should ask for base and exp. value form user. **Note:** Do not use ** operator and math module.

9) Write a program using loop that asks the user to enter an even number. If the number entered is not even then display an appropriate message and ask them to enter a number again. Do not stop until an even number is entered. Print a Congratulatory message at end.

10) Using random module, Simulate tossing a coin N times. Hint: you can use zero for head and 1 for tails.