

# Control Structures

## Lecture 3

Department of Computer Science and Engineering, ITER  
Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar, Odisha, India.



# Contents

- 1 Introduction
- 2 if Conditional Statement
- 3 Iteration (for and while statements)

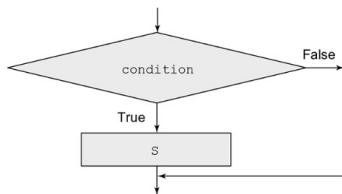
- Control structures are used for non-sequential and repetitive execution of instructions.
- Control statements (also called control structures) are used to control the flow of program execution by allowing non-sequential or repetitive execution of instructions.
- Python supports ***if***, ***for***, ***and*** ***while*** control structures.
- In a control statement, Python code following the colon (:) is indented.

# if Conditional Statement

- *if* statement allows non-sequential execution depending upon whether the condition is satisfied.
- The general form of *if* conditional statement is as follows:  
if *<condition>* :

*Sequence S of statements to be executed*

Here, condition is a Boolean expression which is evaluated at the beginning of the if statement. If condition evaluates to True, then the sequence S of statements is executed, and the control is transferred to the statement following if statement. However, if condition evaluates to False, then the sequence S of statements is ignored, and the control is immediately transferred to the statement.



# if Conditional Statement

- For Example (Marks to be updated to passMarks based on a condition):

```
def moderate(marks, passMarks):  
    """  
    Objective to moderate result by maximum 1 or 2 marks to  
    achieve passMarks  
    Input parameters:  
        marks - int  
        passMarks - int  
    Return Value: marks - int  
    """  
    if marks == passMarks-1 or marks == passMarks-2:  
        marks=passMarks  
    return marks  
  
def main():  
    """  
    Objective: To moderate marks if a student just misses  
    pass marks  
    Input Parameter: None  
    Return Value: None  
    """  
    passMarks=40  
    marks = input("Enter marks:")  
    intmarks = int(marks)  
    moderatedMarks = moderate(intmarks,passMarks)  
    print('Moderated marks:',moderatedMarks)  
  
if __name__=='__main__':  
    main()
```

# if Conditional Statement

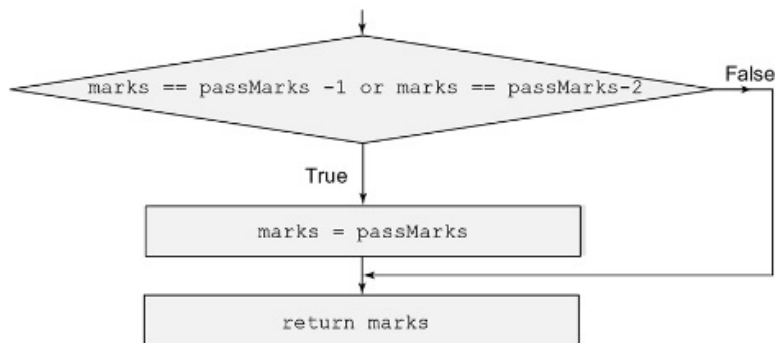
- For Example:

```
>>>
```

Enter marks: 38

Moderated marks: 40

- The flowchart of function moderate is as:



# if Conditional Statement

- The general form of if-else statement is as follows:

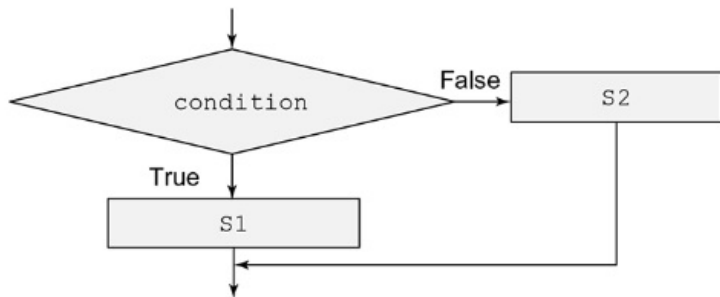
if  $\langle condition \rangle$  :

Sequence S1 of statements to be executed

else:

Sequence S2 of statements to be executed

Here, condition is a Boolean expression. If condition evaluates to True, then the sequence S1 of statements gets executed, otherwise, the sequence S2 of statements gets executed.



# if Conditional Statement

- For Example (Program to authenticate user and allow access to system using if else statement):

```
def authenticateUser(password):  
    """  
    Objective to authenticate user and allow access to system  
    Input parameter : password - string  
    Return Value: message - string  
    """  
    if password == 'magic':  
        message='Login Successful!!\n Welcome to system.'  
    if password != 'magic':  
        message=' Password mismatch!!\n'  
    return message  
  
def main():  
    """  
    Objective: To authenticate user  
    Input Parameter: None  
    Return Value: None  
    """  
    print(' \t LOGIN SYSTEM '  
    print('=====')  
    password = input('Enter Password:')  
    intmarks = int(marks)  
    message = authenticateUser(password)  
    print(message)  
  
if __name__ == '__main__':  
    main()
```

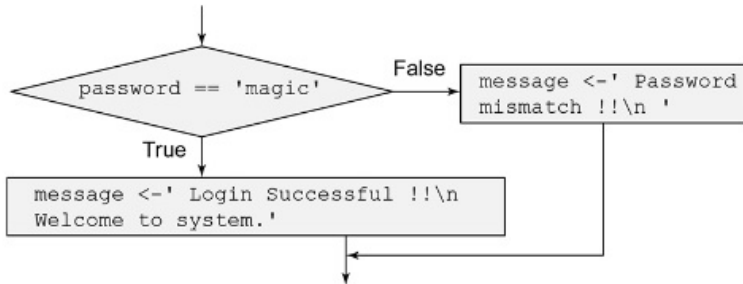


# if Conditional Statement

- For Example:

```
>>>  
if password == 'magic':  
    message = 'Login Successful !! \n'  
else:  
    message = 'Password mismatch !! \n'
```

- The flowchart of if-else statement in the function authenticate User is as:



# if Conditional Statement

- The general form of if-elif-else statement is as follows:

```
if <condition> :  
    Sequence S1 of statements to be executed  
elif <condition2> :  
    Sequence S2 of statements to be executed  
elif <condition3> :  
    Sequence S1 of statements to be executed  
...  
else:  
    Sequence Sn of statements to be executed
```

The clauses elif and else of if control structure are optional. When a control structure is specified within another control structure, it is called nesting of control structures.

# Iteration (for and while statements)

- The process of repetitive execution of statement or a sequence of statements is called a loop.
- Execution of a sequence of statements in a loop is known as an iteration of the loop.
- The control statement *for* is used when we want to execute a sequence of statements a fixed number of times.
- The general form of for statement is as follows:  
for variable in sequence :

Sequence S of Statements

Here, variable refers to the control variable. The sequence S of statements is executed for each value in sequence.

- The function call `range(1, n+1)` generates a sequence of numbers from 1 to n. In general, **`range(start, end, increment)`** produces a sequence of numbers from start up to end (but not including end) in steps of increment. If third argument is not specified, increment is assumed to be 1.

# Iteration (for and while statements)

- For Example (Program to find sum of first n positive integers):

```
def summation(n):  
    """  
    Objective: To find sum of first n positive integers  
    Input Parameter : n - numeric value  
    Return Value: total - numeric value  
    """  
    total = 0  
    for count in range(1,n+1):  
        total += count  
    return total  
  
def main():  
    """  
    Objective: To find sum of first n positive integers based on  
    user input  
    Input Parameter: None  
    Return Value: None  
    """  
    n = int(input('Enter number of terms:'))  
    total = summation(n)  
    print('Sum of first', n, 'positive integers:', total )  
  
if __name__ == '__main__':  
    main()
```

# Iteration (for and while statements)

- A while loop is used for iteratively executing a sequence of statements again and again on the basis of a test-condition.
- The general form of while loop is as follows:

```
while <test – condition> :  
    Sequence S of statements
```

Here, test-condition is a Boolean expression which is evaluated at the beginning of the loop. If the test-condition evaluates to True, the control flows through the Sequence S of statements (i.e, the body of the loop), otherwise the control moves to the statement immediately following the while loop. On execution of the body of the loop, the test-condition is evaluated again, and the process of evaluating the test-condition and executing the body of the loop is continued until the test-condition evaluates to False.

# Iteration (for and while statements)

- For Example (Program to compute sum of numbers):

```
def main():
```

```
    """
```

```
        Objective: To compute sum of numbers entered by user until  
        user provides with null string as the input
```

```
        Input Parameter: None
```

```
        Return Value: None
```

```
    """
```

```
    total=0
```

```
    number = input('Enter a number')
```

```
    while number != '':
```

```
        total += int(number)
```

```
        number = input('Enter a number: ')
```

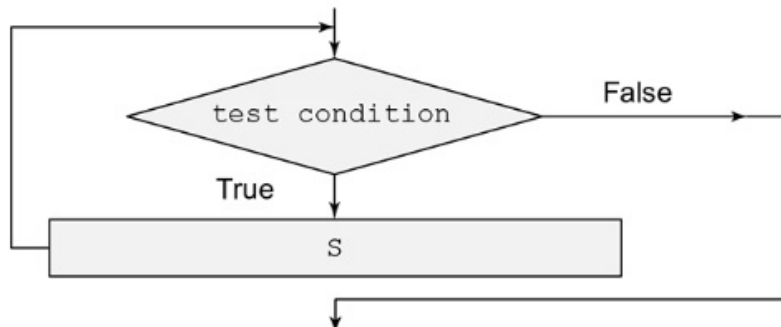
```
    print('Sum of all input numbers is', total)
```

```
if __name__ == '__main__':
```

```
    main()
```

# Iteration (for and while statements)

- The flow diagram of While structure:



# Iteration (for and while statements)

## while statement vs for statement

- Let us consider the code of for statement:

```
for count in range(1, n+1):  
    total += count
```

- We can re-write the above statement in while statement:

```
count = 1  
while count < n+1:  
    total += count  
    count += 1
```

- N.T. - For computing the sum of first few natural numbers, the use of for loop is more elegant and easy as compared to the while loop.



# Iteration (for and while statements)

Example: To print some pictures

- Program to print right triangle and inverted triangle:

```
def rightTriangle(nRows):
```

```
    """
```

```
        Objective: To print right triangle
```

```
        Input Parameter: nRows - integer value
```

```
        Return Value: None
```

```
    """
```

```
    for i in range(1, nRows + 1)
```

```
        print(' * ' * i)
```

```
def invertedTriangle(nRows):
```

```
    """
```

```
        Objective: To print inverted triangle
```

```
        Input Parameter: nRows - integer value
```

```
        Return Value: None
```

```
    """
```

```
    nSpaces = 0
```

```
    nStars = 2 * nRows - 1
```

```
    for i in range(1, nRows+1):
```

```
        print(' ' * nSpaces + ' * ' * nStars)
```

```
        nStars -=2
```

```
        nSpaces +=1
```

# Iteration (for and while statements)

## Example: To print some pictures [Cont.]

```
def main():
```

```
    """
```

Objective: To print right triangle or inverted triangle  
depending on user's choice

Input Parameter: None

Return Value: None

```
    """
```

```
    choice = int(input('Enter 1 for right triangle.\n' + 'Enter 2 for inverted triangle.\n'))
```

```
    assert choice == 1 or choice == 2
```

```
    nRows = int(input('Enter no. of rows'))
```

```
    if choice == 1:
```

```
        rightTriangle(nRows)
```

```
    else:
```

```
        invertedTriangle(nRows)
```

```
if __name__ == '__main__':
```

```
    main()
```

Output: (a) Right Triangle

```
*
* *
* * *
* * * *
* * * * *
```

(b) Inverted Triangle

```
* * * * *
* * * * *
* * * *
* * *
*
```

# Iteration (for and while statements)

## Examples of nested control structures

<pre>for(...):     #Sequence of statements S1 for(...):     #Sequence of statements S2 #Sequence of statements S3</pre>	<pre>while (...):     #Sequence of statements S1     while(...):         #Sequence of statements S2 #Sequence of statements S3</pre>
<pre>for(...):     #Sequence of statements S1     while(...):         #Sequence of statements S2 #Sequence of statements S3</pre>	<pre>while (...):     #Sequence of statements S1     for(...):         #Sequence of statements S2 #Sequence of statements S3</pre>
<pre>for(...):     #Sequence of statements S1 for(...):     #Sequence of statements S2     while(...):         #Sequence of statements S3     #Sequence of statements S4 #Sequence of statements S5</pre>	<pre>while (...):     #Sequence of statements S1     for(...):         #Sequence of statements S2         while(...):             #Sequence of statements S3         #Sequence of statements S4 #Sequence of statements S5</pre>

# Iteration (for and while statements)

## **break, continue and pass statements**

- The break statement is used for exiting from the loop to the statement following the body of the loop.
- The continue statement is used to transfer the control to next iteration of the loop without executing rest of the body of loop.
- The pass statement lets the program go through this piece of code without performing any action.
- The else clause can be used with for and while loop. Statements specified in else clause will be executed on smooth termination of the loop. However, if the loop is terminated using break statement, then the else clause is skipped.

# References

- [1] Python Programming: A modular approach by Taneja Sheetal, and Kumar Naveen, *Pearson Education India, Inc.*, 2017.