FLOW CONTROL

1. DECISION MAKING

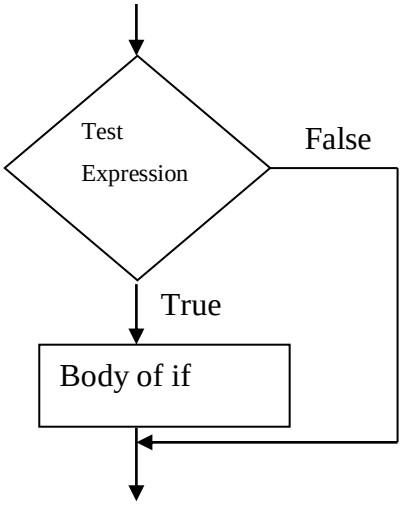
Decision making is required when we want to execute code only if a certain condition is satisfied.

1.1 if statement

Syntax

```
if test expression :
    statement(s)
```

The following shows the chart of *if* statement.



Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True. If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and ends with the first unindented line. Python interprets non-zero values as True. None and 0 are interpreted as False.

Example Program

```
num= int(input("Enter a number: "))
if num= = 0:
    print(“Zero”)
print ("Zero") print ("This is always printed")
```

Output:

```
Enter a number: 0
Zero
This is always printed
```

In the above example, num= = 0 is the test expression. The body of ***if*** is executed only if this evaluates to True. When user enters 0, test expression is True and body inside if is executed. When user enters 2, test expression is False and body inside if is skipped. The statement outside the ***if*** block is shown by unindentation. Hence, it is executed regardless of the test expression or it is always executed.

1.2 if....else statement

Syntax

```
if test expression:
        Body of if
else:
        Body of else
```

The if...else statement evaluates test expression and will execute body of ***if*** only when test condition is True. If the condition is False, body of ***else*** is executed. Indentation is used to separate blocks.

Example Program

```
num= int (input("Enter a number: "))
if num>= 0:
        print("Positive Number Zero")
elase:
        print("Negative Number")
```

Output

```
Enter a number: 5
Positive Number or Zero
```

In the above example, when user enters 5, the test expression is True. Hence the body of ***if*** is executed and body of else is skipped.

1.3 if...elif...else statement

Syntax

```
if test expression:
        Body of if
elif test expression:
        Body of elif
else:
        Body of else
```

The ***elif*** is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. An if block can have only one else block. But it can have multiple elif blocks.

Example Program

```
Num=int (input ("Enter a number:"))
if num > 0:
        print("Positive number")
elif num = = 0:
         print ("Zero")
else:
        print ("Negative number")
```

Output 1

Enter a number: 5

Positive Number

Output 2

Enter a number: -2

Negative Number

1.4 Nested if statement

We can have a *if...elif...else* statement inside another if...elif...else statement. This is called nesting in computer programming. Indentation is the only way to identify the level of nesting.

Example Program

```
num= int (input ("Enter a number:"))
if nun > 0:
        if num= = 0:
                print ("Zero")
        else:
                print("Positive number")
else:
        print("Negative number")
```

Output1

Enter a number: 5

Positive Number

Output 2

Enter a number: -2

Negative Number

2. LOOPS

Generally, statements are executed sequentially. The first statement in a function is execute first, followed by the second, and so on. There will be situations when we need to execute a block of code several number of times. Python provides various control structures that allow for repeated execution. A loop statement allows us to execute a statement or group d statements multiple times.

2.1 for loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or objects that can be iterated. Iterating over a sequence is called traversal.

Syntax

for item in sequence**:**

Body of for

Here, item is the variable that takes the value of the item inside the sequence of each iteration. The sequence can be list, tuple, string, set etc. Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.

Example Program

```
#Program to find the sum of all numbers stored in a list
numbers= [2,4,6,8,10]
sum=0
for item in numbers:
    sum= sum + item
print ("The sum is", sum)
```

Output

The sum is 30

range() function

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing.

len() function is used to find the length of a string or number of elements in a list, tuple, etc.

Example Program

```
flowers = [‘rose', 'lotus', ‘jasmine']
for i in range(len(flowers)):
        print( flowers[i])
```

Output

```
rose
lotus
jasmine
```

We can generate a sequence of numbers using range() function. range (10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step_size as range (start, stop, step_size). The default value of step_size is 1, if not provided. This function does not store all the values in memory. It keeps track of the start, stop, step_size and generates the next number. To make this function to output all the items, we can use the function list ().

Example Program

```
for num in range (2, 10, 2) :
        print(num)
```

Output

```
2
4
6
8
```

enumerate(iterable,start=0)function

This is one of the built-in Python functions. Returns an enumerate object. The parameter iterable must be a sequence, an iterator, or some other object like list which supports iteration.

Example Program

```
flowers= [‘rose’, ‘locus’, ‘jasmine’, ‘sun flower’]
print( list (enumerate(flowers)))
for index, item in enumerate (flowers):
        print (index, item)
```

Output

[(0, rose), (1, lotus), (2, Jasmine), (3, sun flower')]

0 rose

1 lotus

2 jasmine

3 sun flower

2.2. while loop

The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is True. We generally use this loop when we don't know the number of times to iterate in advance.

syntax

while test_expression **:**

Body of while

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process is continued until the test_expression evaluates to False. In Pyth the body of the while loop is determined through indentation. Body starts with indentat and the first unindented line shows the end. When the condition is tested and the result is False, the loop body will be skipped and the first statement after the while loop will executed.

Example Program

```
# Program to find the sum of first N natural numbers
n=int (input("Enter the limit"))
sum=0
i=1
while i<n:
    sum-sum+i
    i= i+1
print ("Sum of first", n, "natural numbers is ", sum)
```

Output

Enter the limit: 5

Sum of first 5 natural numbers is 15

In the above program, when the text Enter the limit appears, 5 is given as the input ie. n=5. Initially, a counter is set to 1 and sum is set to 0. When the condition is checked

for the first time i<=n, it is True. Hence the execution enters the body of while loop. Sum is added with i and i is incremented. Since the next statement is unindented, it assumes the end of while block. This process is repeated when the condition given in the while loop is False and the control goes to the next print statement to print the sum.

while loop with else statement

If the else statement is used with a while loop, the else statement is executed when the condition becomes False. while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is False.

Example Program

```
count=1
while count<=3:
        print("Python Programming")
        count=count +1
else:
        print ("Exit")
print ("End of Program")
```

Output

```
Python Programming
Python Programming
Python Programming
Exit
End of Program
```

Example Program 2

```
count=1
while count<-3:
        print("Python Programming")
        count-count+1
        if count= =2:break
else:
        print ("Exit")
print("End of Program"}
```

Outpul 2

```
Python Programming
```

End of Program

In this example, initially the count is 1. The condition in the while loop is checked a since it is True, the body of the while loop is executed. When the if condition inside while loop is True, ie, when count becomes 2, the control is exited from the while loop. It will not execute the else part of the loop. The control will moves on to the next statement after the else. Hence the print statement End of Program is executed.

3. NESTED LOOPS

Sometimes we need to place a loop inside another loop. This is called nested loop. We can have nested loops for both while and for.

Syntax for nested for loop

```
for iterating_variable in sequence :
    for iterating_variable in sequence:
        statements (s)
    statements(a)
```

Syntax for nested while loop

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

4. CONTROL STATEMENTS

Control statements change the execution from normal sequence. Loops iterate over a block of code until test expression is False, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression. The break and continue statements are used in these cases. Python supports the following three control statements.

1. break
2. continue
3. pass

break statement

The break statement terminates the loop containing it. Control of the program flows to the statement immediately after the body of the loop. If it is inside a nested loop (loop

inside another loop), break will terminate the innermost loop. It can be used with both for and while loops. Fig. 3.6 shows the flow chart of break statement.

Example Program

```
for i in range (2,10,2):
        if i= =6: break
        print (i)
print("End of Program")
```

Output

2

4

End of Program

continue statement

The continue statement is used to skip the rest of the code inside a loop for the current iteration only. Loop does not terminate but continues on with the next iteration. continue returns the control to the beginning of the 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.

Example Program

```
for letter in 'abcd':
        if letter= = 'c': continue
        print (letter)
```

Output

a

b

d

pass statement

In Python programming, pass is a null statement. The difference between a comment and pass statement in Python is that, while the interpreter ignores a comment entirely, pass is not ignored. But nothing happens when it is executed. It results in no operation.

It is used as a placeholder. Suppose we have a loop or a function that is not implemented yet, but we want to implement it in the future. The function or loop

cannot have an empty body. The interpreter will not allow this. So, we use the pass statement to construct a body that does nothing.

Example

```
for val in sequence:
        pass
```

5. TYPES OF LOOPS

There are different types of loops depending on the position at which condition is checked.

5.1 Infinite Loop

A loop becomes infinite loop if a condition never becomes False. This results in a loop that never ends. Such a loop is called an infinite loop. We can create an infinite loop using while statement. If the condition of while loop is always True, we get an infinite loop

Example Program

```
Count=1
while count= =l :
        n=input("Enter a Number:")
        print ("Number", n)
```

This will continue running unless you give CTRL+C to exit from the loop.

5.2 Loops with condition at the top

This is a normal while loop without break statements. The condition of the while loop is at the top and the loop terminates when this condition is False.

5.3 Loop with condition in the middle

This kind of loop can be implemented using an infinite loop along with a condition break in between the body of the loop.

5.4 Loop with condition at the bottom

This kind of loop ensures that the body of the loop is executed at least once. It can be implemented using an infinite loop along with a conditional break at the end. This is similar to the do...while loop in C.

6. Nested List

In Python, we can implement a matrix as nested list (list inside a list). We can treat each element as a row of the matrix. For example X = [[1, 2[, [4, 5], [3, 6]]

would represent a 3x2 matrix. First row can be selected as X10] and the element in first row, first column can be selected as X10][0].

Example Program

```
Matrix=[ [1.2,3],
          [4,5,6]
          [7,8,9] ]
for i in range(len(matrix)):
        for j in range(len(matrix[0])) :
            print (matrix[i][j],end= “    “ )
       print ()
```

Output

1 2 3

4 5 6

7 8 9

7. NESTED DICTIONARIES

Python support dictionaries nested one inside the other. Nesting involves putting a list or dictionary inside another list or dictionary..

Example Program

```
child1= {
                “name”: ”Tom”,
                “year”:2004
        }
child2= {
                “name”: ”Jack”,
                “year”:2006
        }
myfamily= {
                “child1”: child1,
                “child2”: child2
                }
print(myfamily)
```

Output: {'child1': {'name': 'Tom', 'year': 2004}, 'child2': {'name': 'Jack', 'year': 2006}}