## 9. PYTHON – FLOW CONTROL

Table of Contents

9. PYTHON – FLOW CONTROL

1. Flow control

- ✓ The order of statements execution is called as flow of control.

2. Types of the execution

- ✓ Based on requirement the programs statements can executes in different ways like sequentially, conditionally and repeatedly etc.
- ✓ In any programming language, statements will be executed mainly in three ways,
  - o Sequential execution.
  - o Conditional execution.
  - o Looping execution.

Sequential   Conditional (Decision)   Loop (Iteration)
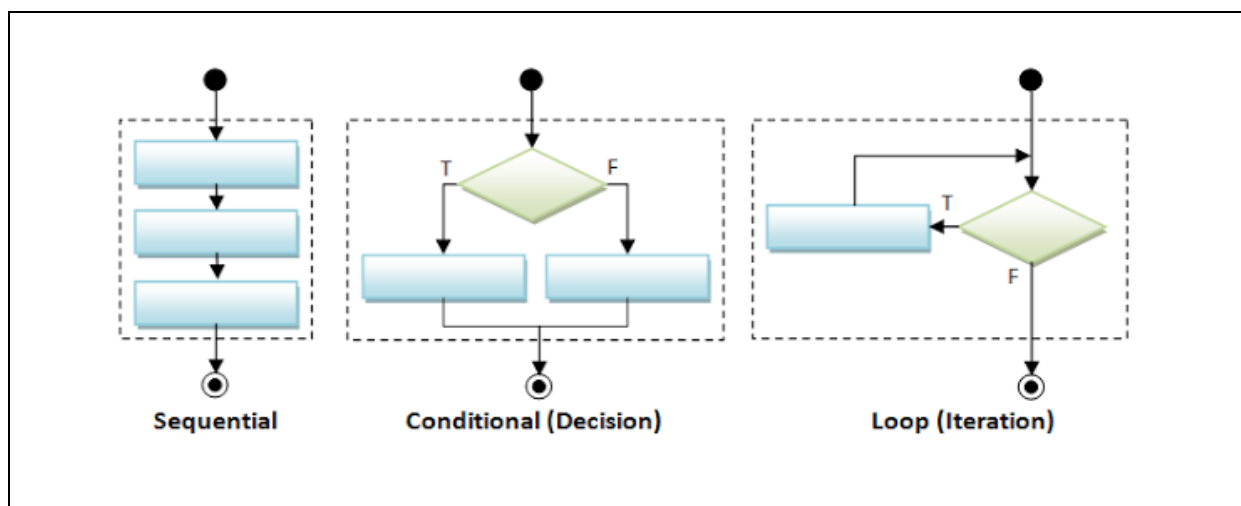
1. Sequential

- ✓ Statements execute from top to bottom, means one by one sequentially.
- ✓ By using sequential statement, we can develop only simple programs.

2. Conditional

- ✓ Based on conditions, statements used to execute.
- ✓ Conditional statements are useful to develop better and complex programs.

3. Looping

- ✓ Based on conditions, statements used to execute randomly and repeatedly.
- ✓ Looping execution is useful to develop better and complex programs.

3. Sequential statements

Program Name   sequential statement: executes from top to bottom
                demo1.py

```
print("one")
print("two")
print("three")
```

output

```
one
two
three
```

2. Conditional or Decision-making statements

- o if   -   valid combination
- o if else   -   valid combination
- o if elif else -   valid combination
- o if elif elif else - valid combination

3. Looping

- ✓ for loop
- ✓ while loop

4. Other keywords

- ✓ break
- ✓ continue
- ✓ pass

4. Indentation

- ✓ Python uses indentation to indicate a block of code.
- ✓ Indentation refers to adding white space before a statement to a particular block of code.
- ✓ Python uses 4 spaces as indentation by default.
    - o However, the number of spaces is up to you, but a minimum of 1 space has to be used.

4.1 IndentationError

- ✓ If we didn't follow the proper indentation then we will get error as **IndentationError**: expected an indented block

5. Conditional or Decision-making statements

5.1 if statement

syntax

if condition**:**

if block statements

out of if block statements

- ✓ **if** is a keyword in python
- ✓ **if** statement contains an expression/condition/value.
- ✓ As per the syntax colon (**:**) is mandatory otherwise it throws syntax error.
- ✓ After **if** statement we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as a bool type, means either **True** or **False**.

- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **if** block statements won't execute.

Program Name	Executing if block statements by using if statement demo2.py
| | ```
x = 1
y = 1

print("x==y value is: ", (x==y))
if x == y:
        print("if block statements executed")
``` |
| output | x==y value is: True
if block statements executed |

Program Name	Executing out of if block statements demo3.py
| | ```
x = 1
y = 2

print("x==y value is: ", (x==y))
if x == y:
        print("if block statements executed")
print("out of if block statements")
``` |
| output | x==y value is: False
out of if block statements |

5.2 if else statement

syntax

if condition**:**
if block statements1

else:
else block statements2

- ✓ **if** and **else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (**:**) is mandatory otherwise it throws syntax error.
- ✓ After **if** and **else** statements we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**

- ✓ If the condition result is **True**, then **if** block statements will be executed
- ✓ If the condition result is **False**, then **else** block statements will be executed.

<table>
<tr><td>Program
Name</td><td>Executing if block statements by using if else statement
demo4.py</td></tr>
</table>

Program Name	Executing if block statements by using if else statement demo4.py

```
x = 1
y = 1

print("x==y value is: ", (x==y))

if x == y:
        print("if block statements executed")
else:
        print("else block statements executed")
```

output	

```
x==y value is: True
if block statements executed
```

Program Name	printing else block statements demo5.py

```
x = 1
y = 2

print("x==y value is: ", (x == y))

if x == y:
        print("if block statements executed")
else:
        print("else block statements executed")
```

output	

```
x==y value is: False
else block statements executed
```

5.3 if elif else statement

---

syntax

    if condition1**:**
        if block statements

    elif condition2**:**
        elif block1statements

    elif condition3**:**
        elif block2statements

    else:
        else block statements

- ✓ **if**, **elif** and **else** are keywords in python
- ✓ **if** statement contains an expression/condition.
- ✓ As per the syntax colon (**:**) is mandatory otherwise it throws error.
- ✓ After **if**, **elif** and **else** statements we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**

- ✓ If the condition result is **True**, then any matched **if** or **elif** block statements will execute.
- ✓ If all **if** and **elif** conditions results are **False**, then **else** block statements will execute.

Make a note

✓ Here, else part is an optional

Program Name	printing corresponding value by using if, elif, else statements demo6.py

```python
print("Please enter the values from 0 to 4")
x = int(input("Enter a number: "))

if x == 0:
        print("You entered:", x)

elif x == 1:
        print("You entered:", x)

elif x == 2:
        print("You entered:", x)

elif x == 3:
        print("You entered:", x)

elif x == 4:
        print("You entered:", x)

else:
        print("Beyond the range than specified")
```

output	

```
Enter a number: 1
You entered: 1

Enter a number: 100
Beyond the range than specified
```

6. Looping

✓ If we want to execute a group of statements in multiple times, then we should go for looping kind of execution.
  o for loop
  o while loop

6.1 for loop

✓ **for** is a keyword in python
✓ Basically, **for** loop is used to get or iterate elements one by one from sequence like string, list, tuple, etc…
✓ While iterating elements from sequence we can perform operations on every element.

Syntax

```
for variable in sequence:
        statements
```

Program Name	Using for loop printing value from list demo7.py values = [10, 20, 30, "Daniel"] for value in values: print(value)
output	10 20 30 Daniel

6.2 while loop

- ✓ **while** is a keyword in python
- ✓ If we want to execute a group of statements repeatedly until the condition reaches to False, then we should go for **while** loop.

Syntax

    Initialization
    while condition**:**
        while block statements
        increment/decrement

- ✓ **while** loop contains an expression/condition.
- ✓ As per the syntax colon (**:**) is mandatory otherwise it throws syntax error.
- ✓ After **while** loop we need to follow indentation otherwise it throws IndentationError.
- ✓ Condition gives the result as bool type, means either **True** or **False**

- ✓ If the condition result is **True**, then while loop executes till the condition reaches to **False**.
- ✓ If the condition result is **False**, then **while** loop execution terminates.

Conclusion

- ✓ Till condition is True the while loop statements will be executed.
- ✓ If the condition reaches to False, then while loop terminate the execution.

Program Name	Printing numbers from 1 to 5 by using while loop demo8.py

```
x = 1
while x <= 5:
        print(x)
        x = x+1


print("End")
```

output	

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

7. break statement

- ✓ **break** is a keyword in python
- ✓ The **break** statement can be used inside the loops.
- ✓ By using break we can break the execution based on some condition.
- ✓ Generally, **break** statement is used to terminate **for** and **while** loops.

Program Name	while loop without break
	 demo9.py
	
	 x = 1
	 while x<=10:
	      print("x=", x)
	      x = x + 1
	
	 print("out of loop")

output	x= 1 x= 2 x= 3 x= 4 x= 5 x= 6 x= 7 x= 8 x= 9 x= 10 out of loop

Program Name	printing just 1 to 5 by using while loop and break demo10.py

```
x = 1
while x <= 10:
        print("x=", x)
        x = x+1
        if x == 5:
                break

print("out of the loop")
```

output

```
x= 1
x= 2
x= 3
x= 4
out of the loop
```

Program Name	break without loop demo11.py

```
x = 1
if x <= 10:
        print(x)
        break
```

output

SyntaxError: 'break' outside loop

8. continue statement

- ✓ **continue** is a keyword in python
- ✓ We can use continue statement to skip current iteration and continue next iteration.

Program Name	continue statement
	demo12.py
	
	``python
	cart = [10, 20, 500, 700, 50, 60]
	for item in cart:
	        if item == 500:
	                continue
	        print(item)
	``

output	10 20 700 50 60

Program Name	continue without loop
	demo13.py
	
	``python
	x = 1
	if x <= 10:
	        continue
	
	print(x)
	``

output	SyntaxError: 'continue' not properly in loop

9. pass statement

- ✓ **pass** is keyword in python.
- ✓ The pass statement is used as a placeholder for future code.
- ✓ It is useful as a placeholder when a statement is required syntactically, but no code needs to be executed.
- ✓ pass is a null operation, when it is executed, nothing happens.
- ✓ We can define an empty function, class, method with pass statement.

Program Name	Function without pass statement demo14.py def upcoming_sales(): upcoming_sales()
output	IndentationError: expected an indented block

Program Name	Function with pass statement demo15.py def upcoming_sales(): pass upcoming_sales()
output	

Make a note

- ✓ We can even define a method or block of code with pass statement.