**Decision making statements**

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.

Following is the general form of a typical decision making structure found in most of the programming languages −



Python programming language assumes any **non-zero** and **non-null** values as TRUE, and if it is either **zero** or **null**, then it is assumed as FALSE value.

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.

In python, decision making is performed by the following statements.

|  |  |
| --- | --- |
| **Statement** | **Description** |
|  |  |
| If Statement | The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed. |
| If - else Statement | The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed. |
| Nested if Statement | Nested if statements enable us to use if ? else statement inside an outer if statement. |

Indentation in Python

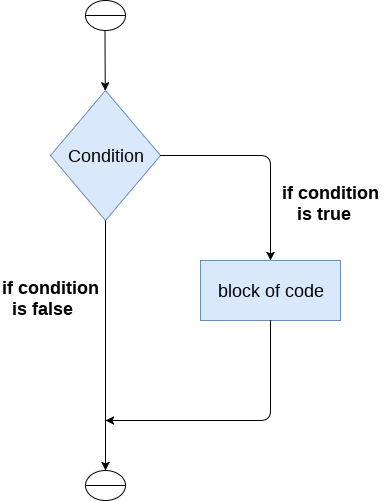
For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block. If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python.

Indentation is the most used part of the python language since it declares the block of code. All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

## The if statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The syntax of the if-statement is given below.

1. **if** expression:
2. statement

### **Example 1**

1. num = int(input("enter the number?"))
2. **if** num%2 == 0:
3. **print**("Number is even")

**Output:**

enter the number?10

Number is even

### **Example 2 : Program to print the largest of the three numbers.**

1. a = int(input("Enter a? "));
2. b = int(input("Enter b? "));
3. c = int(input("Enter c? "));
4. **if** a>b **and** a>c:
5. **print**("a is largest");
6. **if** b>a **and** b>c:
7. **print**("b is largest");
8. **if** c>a **and** c>b:
9. **print**("c is largest");

**Output:**

Enter a? 100

Enter b? 120

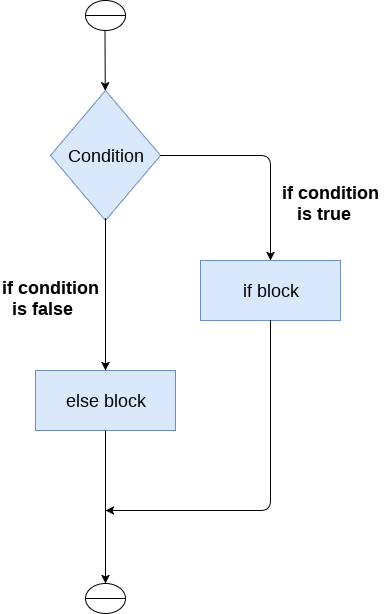
Enter c? 130

c is largest

## if-else statement

The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The syntax of the if-else statement is given below.

1. **if** condition:
2. #block of statements
3. **else**:
4. #another block of statements (else-block)

### **Example 1 : Program to check whether a person is eligible to vote or not.**

1. age = int (input("Enter your age? "))
2. **if** age>=18:
3. **print**("You are eligible to vote !!");
4. **else**:
5. **print**("Sorry! you have to wait !!");

**Output:**

Enter your age? 90

You are eligible to vote !!

### **Example 2: Program to check whether a number is even or not.**

1. num = int(input("enter the number?"))
2. **if** num%2 == 0:
3. **print**("Number is even...")
4. **else**:
5. **print**("Number is odd...")

**Output:**

enter the number?10

Number is even

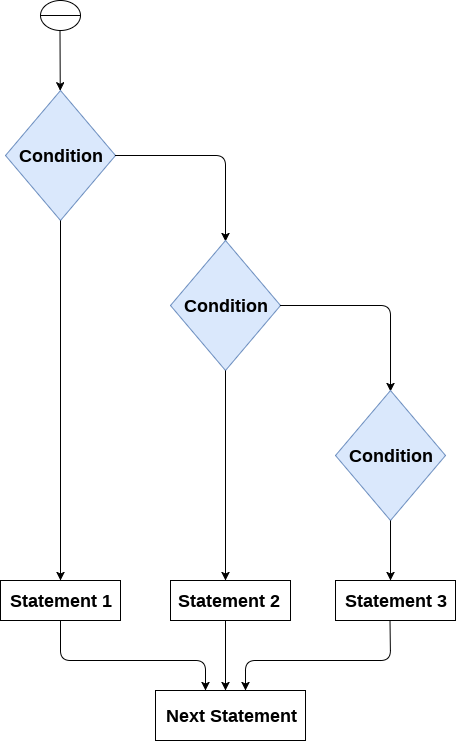
## The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

The syntax of the elif statement is given below.

1. **if** expression 1:
2. # block of statements
4. **elif** expression 2:
5. # block of statements
7. **elif** expression 3:
8. # block of statements
10. **else**:
11. # block of statements



### **Example 1**

1. number = int(input("Enter the number?"))
2. **if** number==10:
3. **print**("number is equals to 10")
4. **elif** number==50:
5. **print**("number is equal to 50");
6. **elif** number==100:
7. **print**("number is equal to 100");
8. **else**:
9. **print**("number is not equal to 10, 50 or 100");

**Output:**

Enter the number?15

number is not equal to 10, 50 or 100

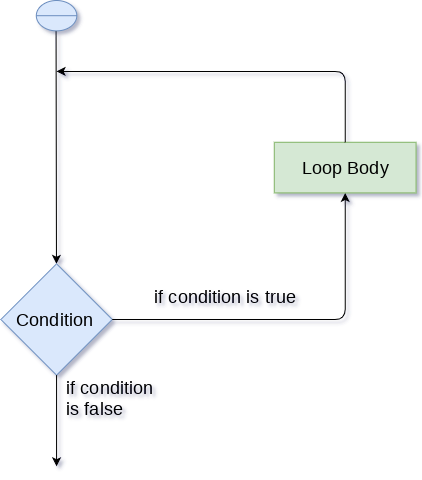
### **Example 2**

1. marks = int(input("Enter the marks? "))
2. f marks > 85 **and** marks <= 100:
3. **print**("Congrats ! you scored grade A ...")
4. lif marks > 60 **and** marks <= 85:
5. **print**("You scored grade B + ...")
6. lif marks > 40 **and** marks <= 60:
7. **print**("You scored grade B ...")
8. lif (marks > 30 **and** marks <= 40):
9. **print**("You scored grade C ...")
10. lse:
11. **print**("Sorry you are fail ?")

# Python Loops

The flow of the programs written in any programming language is sequential by default. Sometimes we may need to alter the flow of the program. The execution of a specific code may need to be repeated several numbers of times.

For this purpose, The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.



## Why we use loops in python?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the print statement 10 times, we can print inside a loop which runs up to 10 iterations.

## Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

There are the following loop statements in Python.

|  |  |
| --- | --- |
| **Loop Statement** | **Description** |
| for loop | The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance. |
| while loop | The while loop is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop. |
| do-while loop | The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs). |

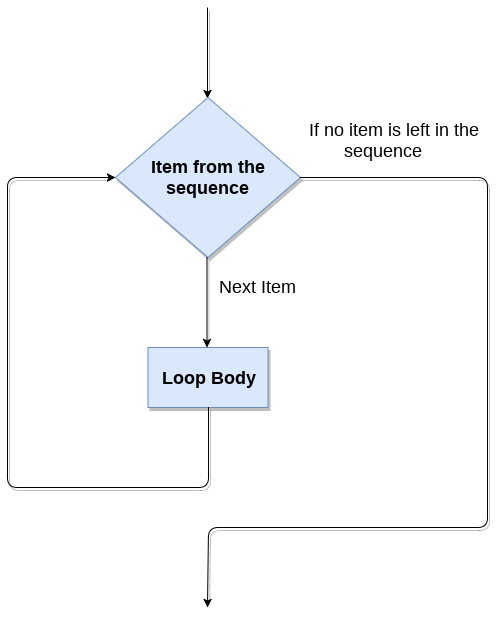
# Python for loop

The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

1. **for** iterating\_var **in** sequence:
2. statement(s)

### **The for loop flowchart**



### **For loop Using Sequence**

**Example-1: Iterating string using for loop**

1. str = "Python"
2. **for** i **in** str:
3. **print**(I)

**Output:**

P

y

t

h

o

n

**Example- 2: Program to print the table of the given number .**

1. list = [1,2,3,4,5,6,7,8,9,10]
2. n = 5
3. **for** i **in** list:
4. c = n\*i
5. **print**(c)

**Output:**

5

10

15

20

25

30

35

40

45

50s

**Example-4: Program to print the sum of the given list.**

1. list = [10,30,23,43,65,12]
2. sum = 0
3. **for** i **in** list:
4. sum = sum+i
5. **print**("The sum is:",sum)

**Output:**

The sum is: 183

### **For loop Using range() function**

**The range() function**

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

**Syntax:**

1. range(start, stop, step size)

* The start represents the beginning of the iteration.
* The stop represents that the loop will iterate till stop-1. The **range(1,5)** will generate numbers 1 to 4 iterations. It is optional.
* The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

**Example-1: Program to print numbers in sequence.**

1. **for** i **in** range(10):
2. **print**(i,end = ' ')

**Output:**

0 1 2 3 4 5 6 7 8 9

**Example - 2: Program to print table of given number.**

1. n = int(input("Enter the number "))
2. **for** i **in** range(1,11):
3. c = n\*i
4. **print**(n,"\*",i,"=",c)

**Output:**

Enter the number 10

10 \* 1 = 10

10 \* 2 = 20

10 \* 3 = 30

10 \* 4 = 40

10 \* 5 = 50

10 \* 6 = 60

10 \* 7 = 70

10 \* 8 = 80

10 \* 9 = 90

10 \* 10 = 100

**Example-3: Program to print even number using step size in range().**

1. n = int(input("Enter the number "))
2. **for** i **in** range(2,n,2):
3. **print**(i)

**Output:**

Enter the number 20

2

4

6

8

10

12

14

16

18

We can also use the **range()** function with sequence of numbers. The **len()** function is combined with range() function which iterate through a sequence using indexing. Consider the following example.

1. list = ['Peter','Joseph','Ricky','Devansh']
2. **for** i **in** range(len(list)):
3. **print**("Hello",list[i])

**Output:**

Hello Peter

Hello Joseph

Hello Ricky

Hello Devansh

## Nested for loop in python

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

**Syntax**

1. **for** iterating\_var1 **in** sequence:  #outer loop
2. **for** iterating\_var2 **in** sequence:  #inner loop
3. #block of statements
4. #Other statements

## Example- 1: Nested for loop

1. # User input for number of rows
2. rows = int(input("Enter the rows:"))
3. # Outer loop will print number of rows
4. **for** i **in** range(0,rows+1):
5. # Inner loop will print number of Astrisk
6. **for** j **in** range(i):
7. **print**("\*",end = '')
8. **print**()

**Output:**

Enter the rows:5

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

### **Example-2: Program to number pyramid.**

1. rows = int(input("Enter the rows"))
2. **for** i **in** range(0,rows+1):
3. **for** j **in** range(i):
4. **print**(i,end = ' ')
5. **print**()

**Output:**

1

22

333

4444

55555

## Using else statement with for loop

Unlike other languages like C, C++, or Java, Python allows us to use the else statement with the for loop which can be executed only when all the iterations are exhausted. Here, we must notice that if the loop contains any of the break statement then the else statement will not be executed.

## Example 1

1. **for** i **in** range(0,5):
2. **print**(i)
3. **else**:
4. **print**("for loop completely exhausted, since there is no break.")

**Output:**

0

1

2

3

4

for loop completely exhausted, since there is no break.

The for loop completely exhausted, since there is no break.

## Example 2

1. **for** i **in** range(0,5):
2. **print**(i)
3. **break**;
4. **else**:**print**("for loop is exhausted");
5. **print**("The loop is broken due to break statement...came out of the loop")

In the above example, the loop is broken due to the break statement; therefore, the else statement will not be executed. The statement present immediate next to else block will be executed.

**Output:**

0

**0**

Python While loop

The Python while loop allows a part of the code to be executed until the given condition returns false. It is also known as a pre-tested loop.

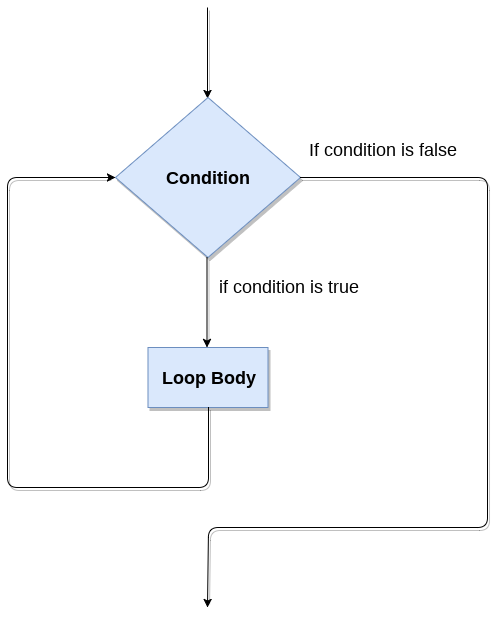
It can be viewed as a repeating if statement. When we don't know the number of iterations then the while loop is most effective to use.

The syntax is given below.

1. **while** expression:
2. statements

Here, the statements can be a single statement or a group of statements. The expression should be any valid Python expression resulting in true or false. The true is any non-zero value and false is 0.

### **While loop Flowchart**



## Loop Control Statements

We can change the normal sequence of **while** loop's execution using the loop control statement. When the while loop's execution is completed, all automatic objects defined in that scope are demolished. Python offers the following control statement to use within the while loop.

**1. Continue Statement -**When the continue statement is encountered, the control transfer to the beginning of the loop. Let's understand the following example.

**Example:**

1. # prints all letters except 'a' and 't'
2. i = 0
3. str1 = 'javatpoint'
5. **while** i < len(str1):
6. **if** str1[i] == 'a' or str1[i] == 't':
7. i =i+1
8. **continue**
9. print('Current Letter :', a[i])
10. i += 1

**Output:**

Current Letter : j

Current Letter : v

Current Letter : p

Current Letter : o

Current Letter : i

Current Letter : n

**2. Break Statement -**When the break statement is encountered, it brings control out of the loop.

**Example:**

1. # The control transfer is transfered
2. # when **break** statement soon it sees t
3. i = 0
4. str1 = 'javatpoint'
6. **while** i < len(str1):
7. **if** str1[i] == 't':
8. i += 1
9. **break**
10. print('Current Letter :', str1[i])
11. i += 1

**Output:**

Current Letter : j

Current Letter : a

Current Letter : v

Current Letter : a

**3. Pass Statement -**The pass statement is used to declare the empty loop. It is also used to define empty class, function, and control statement. Let's understand the following example.

**Example -**

1. # An empty loop
2. str1 = 'javatpoint'
3. i = 0
5. **while** i < len(str1):
6. i += 1
7. pass
8. print('Value of i :', i)

**Output:**

Value of i : 10

### **Example-1: Program to print 1 to 10 using while loop**

1. i=1
2. #The **while** loop will iterate until condition becomes **false**.
3. While(i<=10):
4. print(i)
5. i=i+1

**Output:**

1

2

3

4

5

6

7

8

9

10

### **Example -2: Program to print table of given numbers.**

1. i=1
2. number=0
3. b=9
4. number = **int**(input("Enter the number:"))
5. **while** i<=10:
6. print("%d X %d = %d \n"%(number,i,number\*i))
7. i = i+1

**Output:**

Enter the number:10

10 X 1 = 10

10 X 2 = 20

10 X 3 = 30

10 X 4 = 40

10 X 5 = 50

10 X 6 = 60

10 X 7 = 70

10 X 8 = 80

10 X 9 = 90

10 X 10 = 100

## Infinite while loop

If the condition is given in the while loop never becomes false, then the while loop will never terminate, and it turns into the **infinite while loop.**

Any **non-zero** value in the while loop indicates an **always-true** condition, whereas zero indicates the always-false condition. This type of approach is useful if we want our program to run continuously in the loop without any disturbance.

### **Example 1**

1. **while** (1):
2. print("Hi! we are inside the infinite while loop")

**Output:**

Hi! we are inside the infinite while loop

Hi! we are inside the infinite while loop

### **Example 2**

1. var = 1
2. **while**(var != 2):
3. i = **int**(input("Enter the number:"))
4. print("Entered value is %d"%(i))

**Output:**

Enter the number:10

Entered value is 10

Enter the number:10

Entered value is 10

Enter the number:10

Entered value is 10

Infinite time

## Using else with while loop

Python allows us to use the else statement with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed, and the statement present after else block will be executed. The else statement is optional to use with the while loop. Consider the following example.

### **Example 1**

1. i=1
2. **while**(i<=5):
3. print(i)
4. i=i+1
5. **else**:
6. print("The while loop exhausted")

### **Example 2**

1. i=1
2. **while**(i<=5):
3. print(i)
4. i=i+1
5. **if**(i==3):
6. **break**
7. **else**:
8. print("The while loop exhausted")

**Output:**

1

2

In the above code, when the break statement encountered, then while loop stopped its execution and skipped the else statement.

### **Example-3 Program to print Fibonacci numbers to given limit**

1. terms = **int**(input("Enter the terms "))
2. # first two intial terms
3. a = 0
4. b = 1
5. count = 0
7. # check **if** the number of terms is Zero or negative
8. **if** (terms <= 0):
9. print("Please enter a valid integer")
10. elif (terms == 1):
11. print("Fibonacci sequence upto",limit,":")
12. print(a)
13. **else**:
14. print("Fibonacci sequence:")
15. **while** (count < terms) :
16. print(a, end = ' ')
17. c = a + b
18. # updateing values
19. a = b
20. b = c
22. count += 1

**Output:**

Enter the terms 10

Fibonacci sequence:

0 1 1 2 3 5 8 13 21 34

# Python break statement

The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop.

The break is commonly used in the cases where we need to break the loop for a given condition.

The syntax of the break is given below.

1. #loop statements
2. **break**;

## Example 1

1. list =[1,2,3,4]
2. count = 1;
3. **for** i **in** list:
4. **if** i == 4:
5. **print**("item matched")
6. count = count + 1;
7. **break**
8. **print**("found at",count,"location");

**Output:**

item matched

found at 2 location

## Example 2

1. str = "python"
2. **for** i **in** str:
3. **if** i == 'o':
4. **break**
5. **print**(i);

**Output:**

p

y

t

h

## Example 3: break statement with while loop

1. i = 0;
2. **while** 1:
3. **print**(i," ",end=""),
4. i=i+1;
5. **if** i == 10:
6. **break**;
7. **print**("came out of while loop");

**Output:**

0 1 2 3 4 5 6 7 8 9 came out of while loop

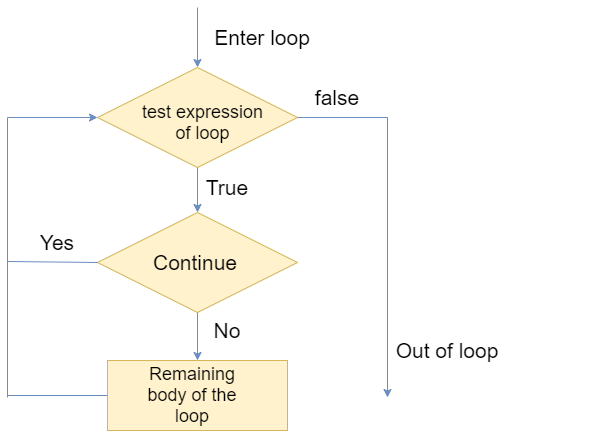
# Python continue Statement

The continue statement in Python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.The continue statement in Python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.

### **Syntax**

1. #loop statements
2. **continue**
3. #the code to be skipped

### **Flow Diagram**



Consider the following examples.

### **Example 1**

1. i = 0
2. **while**(i < 10):
3. i = i+1
4. **if**(i == 5):
5. **continue**
6. **print**(i)

**Output:**

1

2

3

4

6

7

8

9

10

Observe the output of above code, the value 5 is skipped because we have provided the **if condition** using with **continue statement** in while loop. When it matched with the given condition then control transferred to the beginning of the while loop and it skipped the value 5 from the code.

### **Example 2**

1. str = "JavaTpoint"
2. **for** i **in** str:
3. **if**(i == 'T'):
4. **continue**
5. **print**(i)

**Output:**

J

a

v

a

p

o

i

n

t

## Pass Statement

The pass statement is a null operation since nothing happens when it is executed. It is used in the cases where a statement is syntactically needed but we don't want to use any executable statement at its place.

For example, it can be used while overriding a parent class method in the subclass but don't want to give its specific implementation in the subclass.

Pass is also used where the code will be written somewhere but not yet written in the program file. Consider the following example.

### **Example**

1. list = [1,2,3,4,5]
2. flag = 0
3. **for** i **in** list:
4. **print**("Current element:",i,end=" ");
5. **if** i==3:
6. **pass**
7. **print**("\nWe are inside pass block\n");
8. flag = 1
9. **if** flag==1:
10. **print**("\nCame out of pass\n");
11. flag=0

**Output:**

Current element: 1 Current element: 2 Current element: 3

We are inside pass block

Came out of pass

Current element: 4 Current element: 5

**References:**

https://www.tutorialspoint.com/python/python\_decision\_making.htm