CONDITIONAL EXECUTION

To write useful programs, it is needed to check the condition and change the behaviour of the program accordingly, Conditional statements give this ability. They allow executing statements selectively based on certain decisions. such type of decision control statements are Known as selection control statements or conditional branching statements. Types are ...

- Conditional Execution (if)

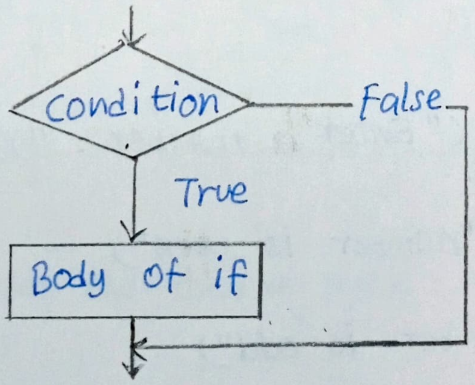
The if selection structure performs an indicated action only when the condition is True, otherwise the action is skipped. In python, the body of the if statement is indicated by the indentation.

* Syntax :

```
if <condition>:
        statements(s)
statement(x)
```

- The colon :
- The indentation are must.

* Flowchart :



* Code :

```
num = 3
if num > 0 :
    print(num, " is a positive number.")
```

Output :

```
3 is a positive number
```

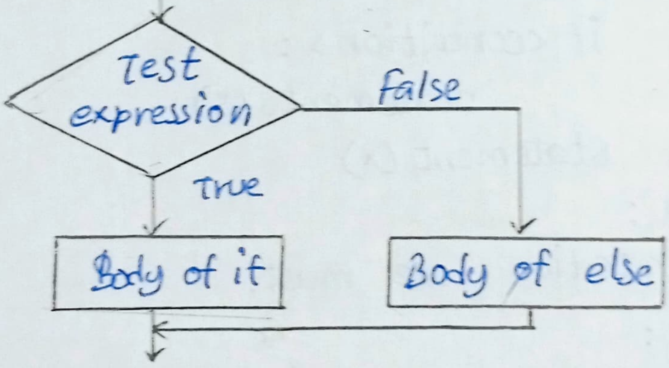
- Alternative Execution (if - else)

The if-else statement evaluates the condition and executes the true statement block only when the condition is true. If the condition is False, false statement block is executed. Indentation is used to separate the blocks.

* Syntax :

```
if <condition>:
        Body of if
else :
     Body of else
Statement (x)
```

* Flowchart :



*. Code :

```
num = int (input (" Enter a number : "))
if num % 2 == 0 :
        print ("number is even")
else :
    print ("number is odd")
```

Output:

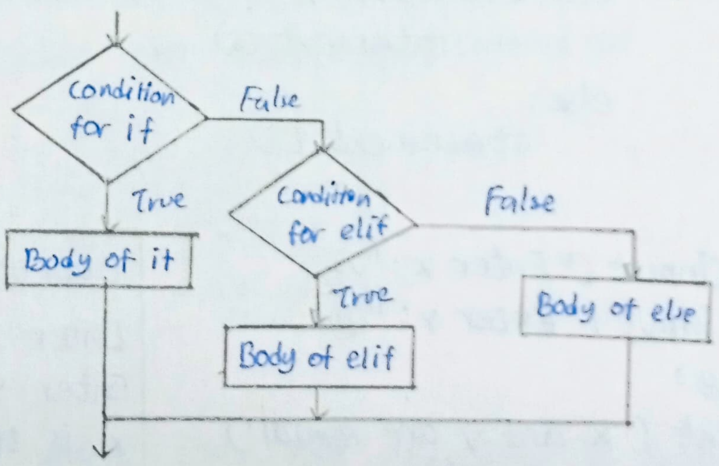
```
Enter a number : 8
number is even
Enter a number : 9
number is odd
```

• Chained Conditional execution (if - elif - else)

Sometimes there are more than two posibilities and need more than two branches. One way to express a computation like that is a chained conditional execution. The elif statement allows checking multiple expressions for truth value and executing a block of code as soon as one of the conditions evaluates to true.

* syntax :
```
if <condition-1> :
        Body of if
elif <condition-2> :
        Body of elif
          :
else :
        Body of else
```

*. Flowchart :



* Code :
```
avg = input ("Enter the average mark : ")
if avg > 80 :
      print ("Grade is A")
elif avg > 70 :
      Print ("Grade is B")
elif avg > 60 :
      Print (" Grade is c")
else :
      print (" Grade is U")
```

• Nested conditionals

There may be a situation to check for another condition after a condition resolves to true. In such a situation, the nested if construct can be used. One conditional can also be nested within another. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting.

* syntax :

```
if <condition-1> :
        statement (s)
        if <condition-2> :
                statement (s)

              : : :
        else :
                statement (s)
elif <condition- n> :
        statement (s)
else:
    statement (s)
```

* Code :

```
x = int (input ("Enter x :"))
y = int (input ("Enter y: "))
if x == y :
        print ('x and y are equal')
else :
    if x < y :
        Print ('x is less than y')
    else :
        print ('x is greater than y')
```

Output :

```
Enter x: 56
Enter y : 98
x is less than y

Enter x : 23
Enter x : 23
x and y are equal
Enter x : 94
Enter y : 47
x is greater than y
```

ITERATION

Looping is also called as repetition structure or iteration. An iteration structure allows the programmer to perform an action which is to be repeated until the given condition is True.

- State

An iteration involves state variables which keep track of the state of each iteration variable for each pass through the iteration. An iteration table is a way to visualize the state of the iteration.

- The columns of the iteration table are the state variables involved in the iteration and the rows are the values of the state variables in each invocation of the iteration.
- Example : Given a list L1 = [1,2,3,4] the iteration table for reverse (L1) would be

List	Result
[1,2,3,4]	[]
[2,3,4]	[1]
[3,4]	[2,1]
[4]	[3,2,1]
[]	[4,3,2,1]

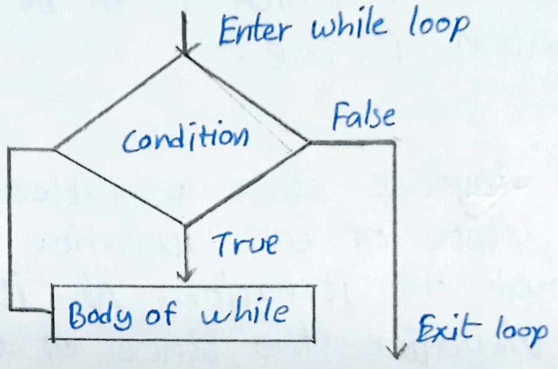
- While loop

In the while loop, the condition is tested before any of the statement block is executed. If the condition is True, Then only the statements will be executed. If the condition is false the control will jump the immediate statement outside the while loop. The while loop will executed as long as the condition is true. In python, the body of the while loop is determined through indentation. It is used if the number of times to iterate is not known before.

* Syntax :

```
while <conditions> :
        Body of while
statement - x
```

*. Flowchart :



*. Code : # sum = 1 + 2 + 3 + ... + n

```
n = int (input (" Enter n: "))
sum = 0
i = 1
while i<= n :
        sum = sum + i
        i = i + 1
Print ("The sum is", sum)
```

Output:

```
Enter n : 10
The sum is 55
```

- for Loop

The for statement is used to iterate over a range of values or a sequence. Iterating over a sequence is called traversal.

The for loop is executed for each of the items in the range. These values can be numeric, string, list, or tuple.

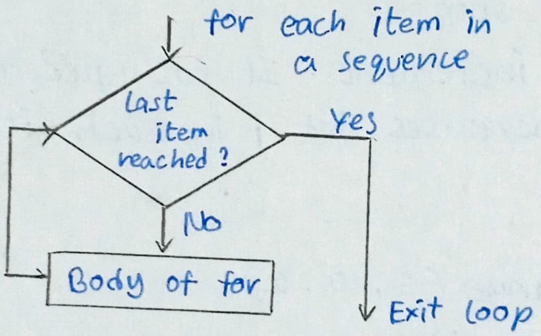
With every iteration of the loop, the control variable checks whether each of the values in the range have

been traversed or not.
when all the items in the range are exhausted, the statements within loop are not executed; the control is then transferred to the statement immediately following the for loop.
The number of iterations or times the loop will execute is known in advance while using for loop.

* Syntax :

```
            for val in sequence :
                    body of for
            Statement(x)
```

* Flowchart :



* Code :

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
sum = 0

for val in numbers :
        sum += val
print ("The sum is", sum)
```

Output :

```
The sum is 48
```

using range() Function

The python has the built-in function range(), which is most often used in for loops. It is used to create a list containing a sequence of integers from the given start value up to stop value (exculding stop value), with a difference of the given step value. The arguments must be integers.

*. Syntax :

```
range ( [start,] stop [, step])
```

Here,

- start and step are optional (step parameter can be positive or a negative integer excluding zero.
- If the start argument is omitted, it defaults to 0,
- Stop is the last no of range. The excat end position is, stop-1.
- Step is the increment. If not specified, by default the value increases by 1 in each iteration.

* Code :

```
for i in range (5, 10, 2):
        print (i)
```

Output :

```
5
7
9
```

~ Thank you