2. Flow Control statements:--------->

- Flow control determines the order in which the statements are executed at runtime.
- They are categorized into conditional statements(if), iterating or looping statements(while and for) and finally transfer statements(break, continue, pass, return).

2. Flow Control Statements: Overview

2.1. Conditional statements(if)

- ***2.1.1.*** if
- ***2.1.2.*** else
- ***2.1.3.*** if else elif ladder

2.2. Iterating or looping statements(while and for)

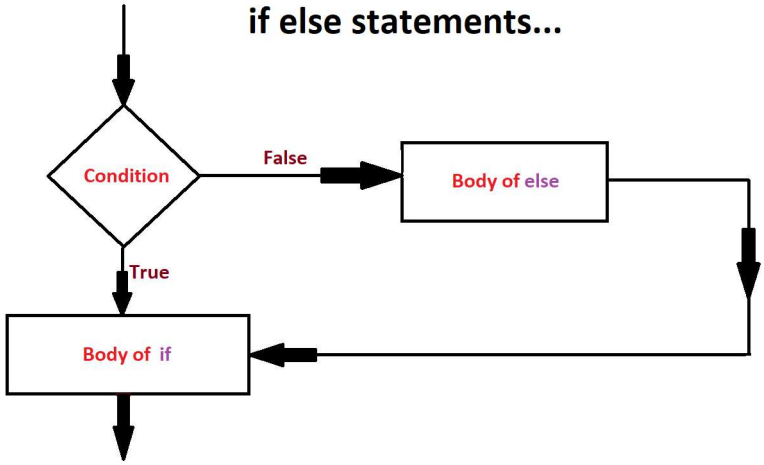
- ***2.2.1.*** while loop(conditional looping)
- ***2.2.2.*** for loop(counting looping)

2.3. Transfer statements(break, continue, pass, return)

- ***2.3.1.*** break
- ***2.3.2.*** continue
- ***2.3.3.*** pass
- ***2.3.4.*** return

2.1. Conditional statements:(if)

- You can use if statements to run code if a certain condition holds.
- If an expression evaluates to True, some statements are carried out. Otherwise, they aren't carried out.
- An if statement looks like this:
  if expression:
  > statements
- Python uses indentation (white space at the beginning of a line) to delimit blocks of code. Other languages, such as C, use curly braces to accomplish this, but in Python indentation is mandatory; programs won't work without it. As you can see, the statements in the if should be indented.



Syntax of if statement:

```
if (test expression) :
    statements
        .
        .
        .
        .
    statements
```

LearnFluent

In [1]:
```
# Example:

if(10 > 5):
    print("10 is greater than 5.")
print("Program ended.")
```

```
10 is greater than 5.
Program ended.
```

The expression determines whether 10 is greater than five. Since it is, the indented statement runs, and "10 greater than 5" is output. Then, the unindented statement, which is not part of the if statement, is run, and "Program ended" is displayed.

Notice the colon at the end of the expression in the if statement.

Else

An else statement follows an if statement, and contains code that is called when the if statement evaluates to False.

As with if statements, the code inside the block should be indented.

Syntax of if else statements:

```
if (test expression) :
    statements
        .
        .
    statements
else :
    statements
        .
        .
    statements
```

LearnFluent

In [2]:
```
# Example:
x = 4
if x == 5:
    print("Yes")
else:
    print("No")
```

```
No
```

In [3]:
```
"""
Question:
Write a program to find that the number is even or odd.
"""

n = int(input("Enter a nuimber to check even or odd: "))
if(n % 2 == 0):
    print(n, "is Even no.")
else:
    print(n, "is Odd no.")
```

```
Enter a nuimber to check even or odd: 15
15 is Odd no.
```

In [4]:
```
"""
Question:
WAP to get the grades according to the marks.
91-100--->A
81-90---->B
71-80---->C
61-70---->D
51-60---->E
00-50---->Fail
"""
# solution 1: ( using only if statement )

marks = int(input("Enter the marks: "))
if(marks >= 91 and marks <= 100):
    print("Grade is A")
if(marks >= 81 and marks <= 90):
    print("Grade is B")
if(marks >= 71 and marks <= 80):
    print("Grade is C")
if(marks >= 61 and marks <= 70):
    print("Grade is D")
if(marks >= 51 and marks <= 60):
    print("Grade is E")
if(marks >= 0 and marks <= 50):
    print("You are Fail")
```

```
Enter the marks: 87
Grade is B
```

Syntax of if else ladder:
(nested if else)

```
if (test expression 1) :
    statements
else :
    if (test expression 2) :
        statements
    else :
        if (test expression 3):
            statements
        else :
            if (test expression 4) :
                statements
            else :
                if (test expression 5) :
                    statements
                else :
                    statements
```

LearnFluent

In [5]:
```
# solution 2: ( using if else statements)   -----> if else ladder

marks = int(input("Enter the marks: "))
if(marks >= 91 and marks <= 100):
    print("Grade is A")
else:
    if(marks >= 81 and marks <= 90):
        print("Grade is B")
    else:
        if(marks >= 71 and marks <= 80):
            print("Grade is C")
        else:
            if(marks >= 61 and marks <= 70):
                print("Grade is D")
            else:
                if(marks >= 51 and marks <= 60):
                    print("Grade is E")
                else:
                    if(marks >= 0 and marks <= 50):
                        print("You are Fail")
                    else:
                        print("Marks is invalid")
```

```
Enter the marks: 77
Grade is C
```

Syntax of if elif else ladder:

```
if (test expression 1) :
    statements
        .
elif (test expression 2) :
    statements
        .
elif (test expression 3) :
    statements
        .
elif (test expression 4) :
    statements
        .
else :
    statements
        .
        .
```

LearnFluent

In [6]:
```
# solution 3: ( using if, elif and else statements)    ------> (if elif  else lad

marks = int(input("Enter the marks: "))
if(marks >= 91 and marks <= 100):
    print("Grade is A")
elif(marks >= 81 and marks <= 90):
    print("Grade is B")
elif(marks >= 71 and marks <= 80):
    print("Grade is C")
elif(marks >= 61 and marks <= 70):
    print("Grade is D")
elif(marks >= 51 and marks <= 60):
    print("Grade is E")
elif(marks >= 0 and marks <= 50):
    print("You are Fail")
else:
    print("Marks is invalid.")
```

```
Enter the marks: 67
Grade is D
```

In [7]:
```
"""
Question:
WAP to find the maximum of three numbers.
"""

a, b, c = map(int, input("Enter the three numbers seperated by space: ").split())
if(a > b):
    if(a > c):
        print(a, "is maximum.")
    else:
        print(c, "is maximum.")
else:
    if(b > c):
        print(b, "is maximum.")
    else:
        print(c, "is maximum.")
```

```
Enter the three numbers seperated by space: 12 14 18
18 is maximum.
```

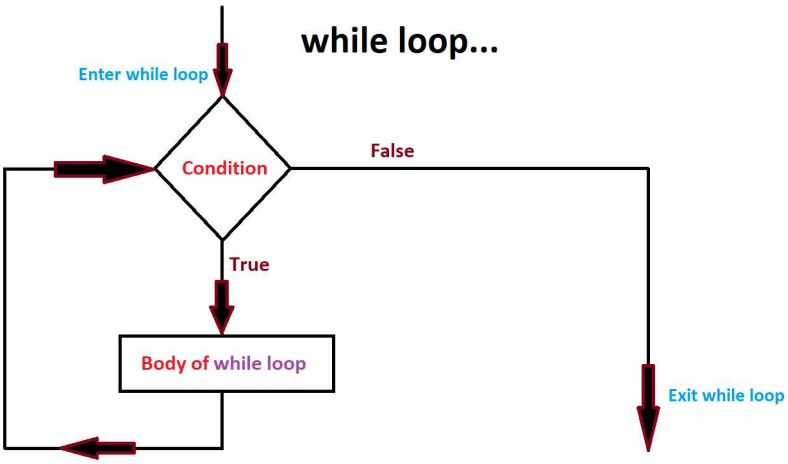
In [ ]:
```

```

2.2. Iterating or looping statements:

2.2.1. While loop: (conditional looping)

- An if statement is run once if its condition evaluates to True, and never if it evaluates to False.
- A while statement is similar, except that it can be run more than once. The statements inside it are repeatedly executed, as long as the condition holds. Once it evaluates to False, the next section of code is executed.
- Below is a while loop containing a variable that counts up from 1 to 5, at which point the loop terminates.

terminates.



Synatx of while loop:

```
while (boolean test expression) :
    statements
        .
        .
        .
    statements
```

LearnFluent

In [8]:
```
i = 1
while(i <= 5):
    print(i)
    i = i + 1

print("Finished!")
```

```
1
2
3
4
5
Finished!
```

- The code in the body of a while loop is executed repeatedly. This is called iteration.

In [9]:
```
"""
Question:
WAP to print the counting number up to 20. (1 to 20)
"""

i = 1
while(i < 21):
    print(i)
    i += 1
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

- The **infinite loop** is a special kind of while loop; it never stops running. Its condition always remains True.
- An example of an infinite loop:

In [10]:
```
# This program would indefinitely print "In the loop".
while(1 == 1):
    # print("In the loop")
    # infinite loop
    break
```

- You can stop the program's execution by using the Ctrl-C shortcut or by closing the program.

In [11]:
```
"""
Question:
WAP to print odd numbers between x and y.
"""

x = int(input("Enter the min. number: "))
y = int(input("Enter the max. number: "))
i = x
if(i % 2 == 0):
    i = i + 1
while(i < y):
    print(i)
    i += 2
```

```
Enter the min. number: 12
Enter the max. number: 32
13
15
17
19
21
23
25
27
29
31
```

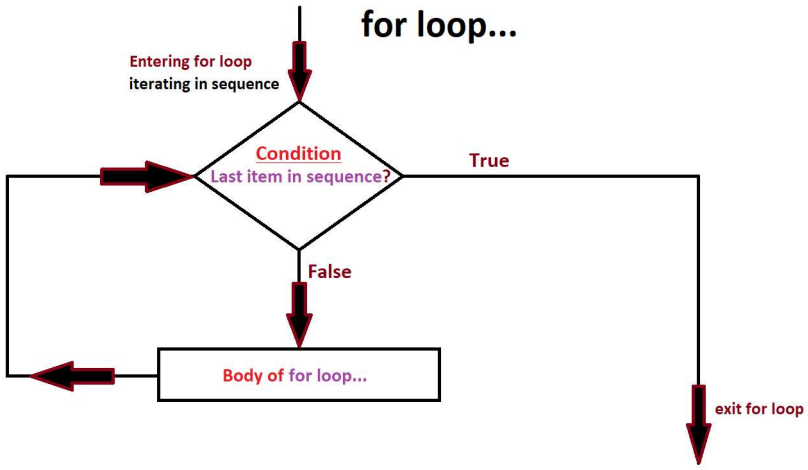
In [12]:
```
"""
Question:
WAP to print
"""
```

Out[12]:
```
'\nQuestion:\nWAP to print\n'
```

In [ ]:
```

```

2.2.2. for loop: (Counting looping)

- Sometimes, you need to perform code on each item in a sequence. This is called iteration, and it can be accomplished with a while loop and a counter variable.



LearnFluent

Syntax of for loop:

Examples of Sequences: **list, tuple, string, range etc**
dictionary items.
sets

Syntax:

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

LearnFluent

In [13]:
```
# Example:

words = ["hello", "world", "spam", "eggs"]
counter = 0
max_index = len(words) - 1

while counter <= max_index:
    word = words[counter]
    print(word + "!")
    counter = counter + 1
```

```
hello!
world!
spam!
eggs!
```

- The example above iterates through all items in the list, accesses them using their indices, and prints them with exclamation marks.

- Iterating through a list using a while loop requires quite a lot of code, so Python provides the for loop as a shortcut that accomplishes the same thing.
- The same code from the previous example can be written with a for loop, as follows:

- iterate over the elements of the sequences like list, tuple, string, set, range etc

In [14]:
```
words = ["hello", "world", "spam", "eggs"]
for word in words:
    print(word + "!")
```

```
hello!
world!
spam!
eggs!
```

- The for loop in Python is like the foreach loop in other languages.

- The for loop is commonly used to repeat some code a certain number of times. This is done by combining for loops with range objects.

In [15]:
```
for i in range(5):
    print("Hello!")
```

```
Hello!
Hello!
Hello!
Hello!
Hello!
```

In [16]:
```
for i in range(0, 20, 2):
    print(i)
```

```
0
2
4
6
8
10
12
14
16
18
```

In [17]:
```
"""
Question:
WAP to print the counting number from 50 to 70.
"""

for i in range(50, 71, 1):
    print(i)
```

```
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
```

In [18]:
```
"""
Question:
WAP to get the product of the numbers in the list.
"""

l = list(map(int, input("Enter the numbers in the list seperated by space: ").spl
product = 1
for x in l:
    product *= x
print(product)
```

```
Enter the numbers in the list seperated by space: 1 2 3 4 5
120
```

In [19]:
```
"""
Question:
WAP to print the multiplication table of a given number.
"""

n = int(input("Enter the number: "))
for j in range(1, 11):
    print(j * n)
```

```
Enter the number: 5
5
10
15
20
25
30
35
40
45
50
```

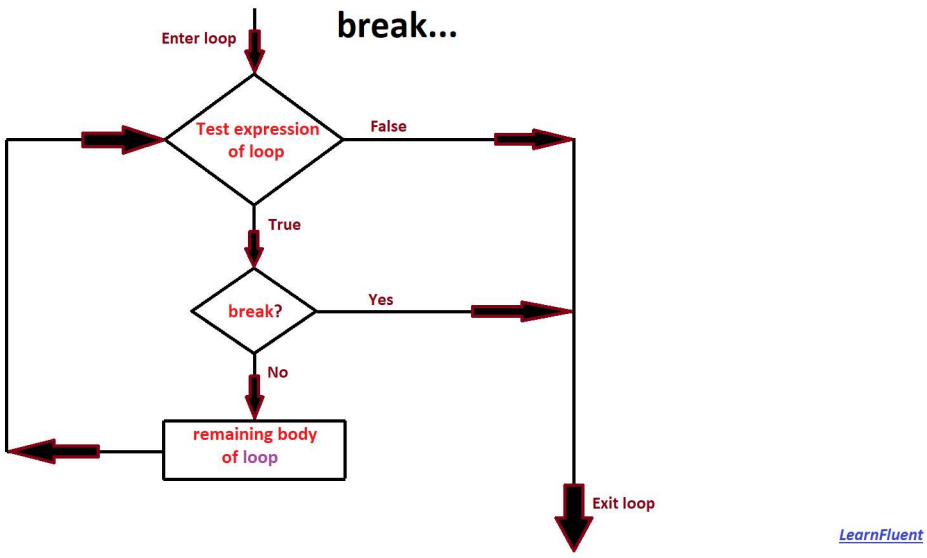
In [ ]:

2.3. break, continue, pass and return: (Transfer statements)

In [ ]:

2.3.1. break:

- To end a while loop prematurely, the break statement can be used.
- When encountered inside a loop, the break statement causes the loop to finish immediately.



In [20]:
```
lst = [3, 6, 9, 5, 12]

for i in lst:
    if(i == 5):
        break
    print(i)
print("Finished")
```

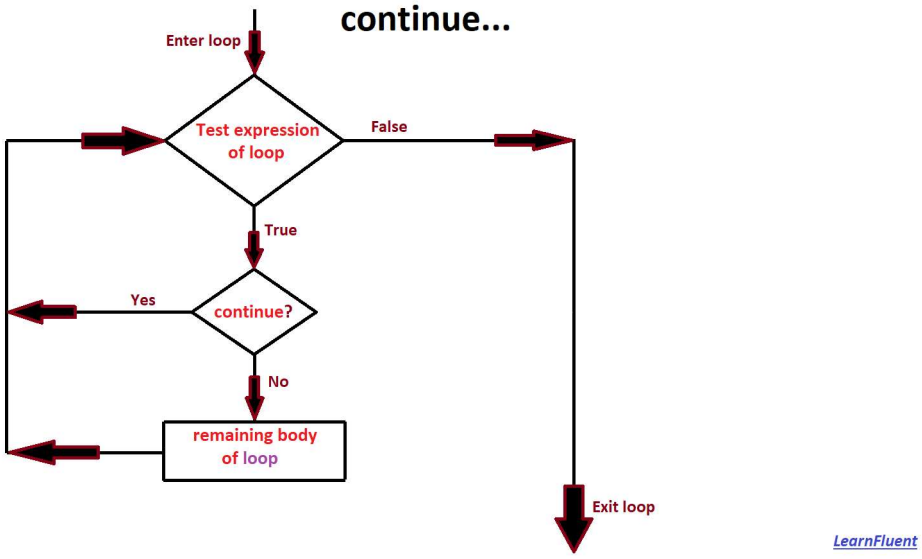
```
3
6
9
Finished
```

- Using the break statement outside of a loop causes an error.

In [ ]:
```

```

2.3.2. continue:

- Another statement that can be used within loops is continue.
- Unlike break, continue jumps back to the top of the loop, rather than stopping it.



In [21]:
```
i = 0
while True:
    i += 1
    if(i == 2):
        print("Skipping 2")
        continue
    if(i == 5):
        print("Breaking")
        break
    print(i)

print("Finished")
```

```
1
Skipping 2
3
4
Breaking
Finished
```

- Basically, the continue statement stops the current iteration and continues with the next one.
- Using the continue statement outside of a loop causes an error.

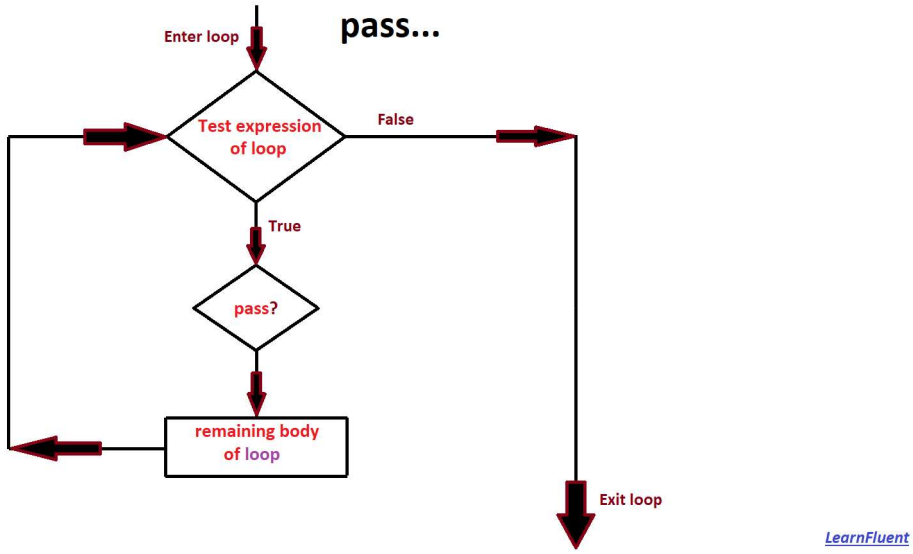
In [22]:
```
"""
Question:
WAP to print from 1 to 20 and skip the multiples of 3.
"""

x = 0
while(x < 20):
    x += 1
    if(x % 3 == 0):
        continue
    print(x, end = "  ")
```

```
1  2  4  5  7  8  10  11  13  14  16  17  19  20
```

In [ ]:
```

```

2.3.3. pass:

- Pass statement is used to do nothing. It can be used inside a loop or if statement to represent no operation. Pass is useful when we need statements syntactically correct but we do not want to do any operation.



In [23]:
```
# we can cover this part(assertion) in exceptional handling.
```

2.3.4. assert:

- An assertion is a sanity-check that you can turn on or turn off when you have finished testing the program.
- An expression is tested, and if the result comes up false, an exception is raised.
- Assertions are carried out through use of the assert statement.

In [24]:
```
print(1)
assert 2 + 2 == 4
print(2)
assert 1 + 1 == 3
print(3)
```

```
1
2

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-24-98aeb3b8a49e> in <module>
      2 assert 2 + 2 == 4
      3 print(2)
----> 4 assert 1 + 1 == 3
      5 print(3)

AssertionError:
```

- Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.

- The assert can take a second argument that is passed to the AssertionError raised if the

assertion fails.

In [25]:
```
temp = -10
assert (temp >= 0), "Colder than absolute zero!"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-25-b5d283b4e7fd> in <module>
      1 temp = -10
----> 2 assert (temp >= 0), "Colder than absolute zero!"

AssertionError: Colder than absolute zero!
```

In [26]:
```
x = int(input("Enter a number greater than 10: "))
assert x > 10, "Wrong number entered"
print("You Entered ", x)
```

```
Enter a number greater than 10: 5

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-26-50d366c10450> in <module>
      1 x = int(input("Enter a number greater than 10: "))
----> 2 assert x > 10, "Wrong number entered"
      3 print("You Entered ", x)

AssertionError: Wrong number entered
```

- AssertionError exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, this type of exception will terminate the program.

In [ ]:
```
```

In [19]:
```
# we will study return in upcoming modules.
```