Conditional Statements/ Decision Making Statements

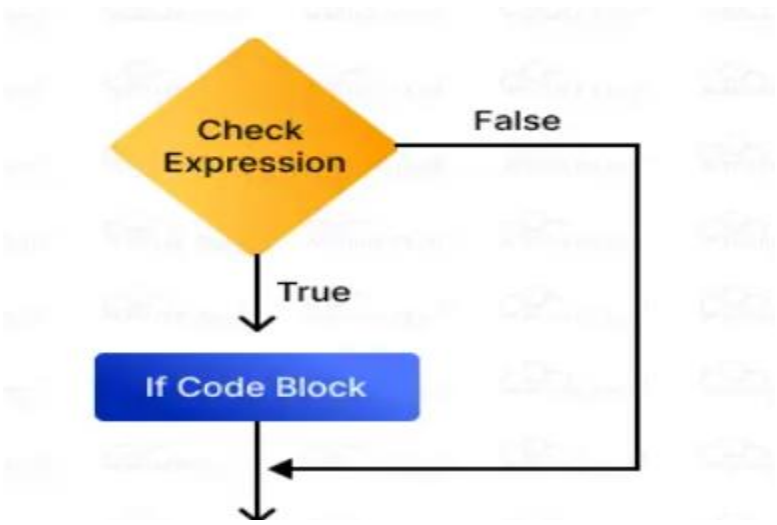
Flow of control

The flow of python interpreter is usually sequential. Where it starts from first and executes each line one by one and then ends execution. But there might be certain instances where a set of lines need to be executed only if a certain condition is true/false. As these statements control the flow of program these are called control statements/conditional statements.

Conditional statements in Python are used to execute certain blocks of code based on specific conditions. These statements help control the flow of a program, making it behave differently in different situations.

If Statement

If statement is the simplest form of a conditional statement. It executes a block of code if the given condition is true.




```
# Syntax -
# if condition:
    # Statements to execute if
    # condition is true
```

```
age=9 # declaring of variable
# it will execute the condition if the evaluated condition is true
# if the condition is false, it will do nothing
```

```
# proper indentation is required so interpreter will get to know that it is block
of if statement
if age>18:
    print("You are eligible") # this is the block of if statement

if False:
    # Here False is 0
    print("This will never get execute coz False value is 0 means false")
print("This is out of if block/statement") # This part always get execute coz
this is not part of if block.
# No effect on this print statement whether if value is true or false.

if True:
    print("This is always True, so if block will get execute")
print("This is out of if block/statement")
```

[Note: A condition is checked, if it is true the statements under if gets executed, if it is false then it comes out of the conditional statement.]

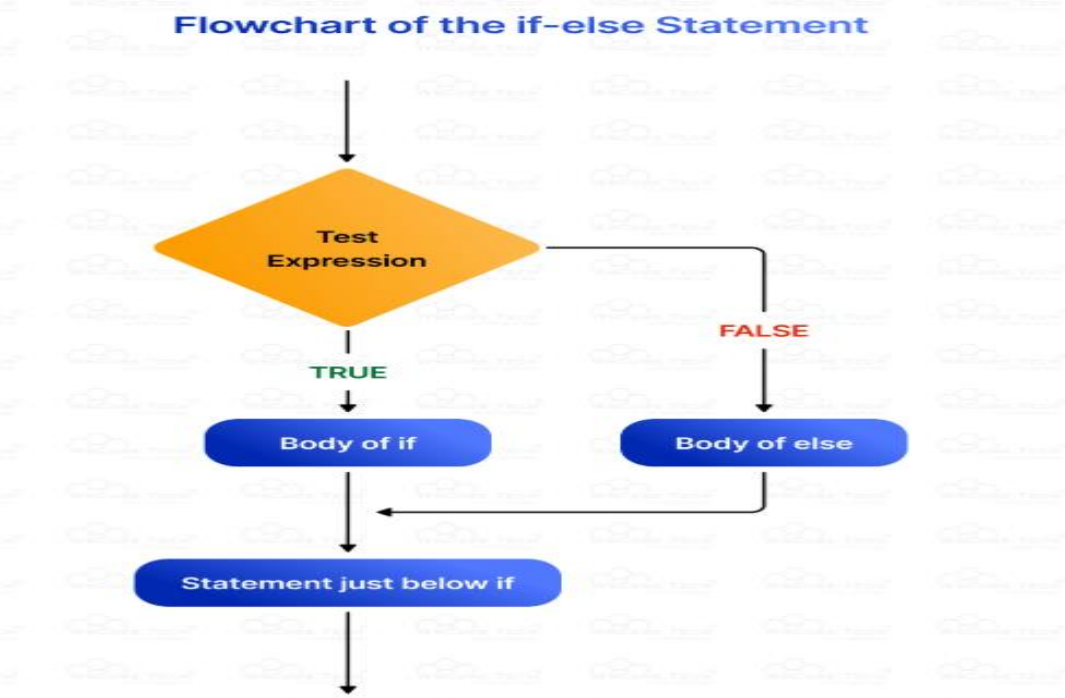
Short Hand if Statement

Short-hand if statement allows us to write a single-line if statement.

```
# Syntax:
# if condition: statement
if age>18: print("You are eligible")
```

If else Statement

It allows us to specify a block of code that will execute if the condition(s) associated with an if or elif statement evaluates to False. Else block provides a way to handle all other cases that don't meet the specified conditions.




```
# Syntax of if-else:
# if condition:
    # code to execute if condition is true

# else:
    # code to execute if condition is false

if age>18:
 print("You are eligible")
else:
    print("You are not eligible")
```

Short Hand if-else

The short-hand if-else statement allows you to write a single-line if-else statement.

```
# Syntax:
# statement_if_true if condition else statement_if_false
print("You are eligible") if age>20 else print("You are not eligible")
```

elif Statement

elif statement in Python stands for "else if." It allows us to check multiple conditions , providing a way to execute different blocks of code based on which condition is true. Using elif statements makes our code more readable and efficient by eliminating the need for multiple nested if statements.

```
# Syntax:
# if condition1:
     # code to execute if condition1 is true
# elif condition2:
     # code to execute if condition2 is true
# else:
    # code to execute if none of the above conditions are true

age=int(input("Enter your age"))
if age>20 and age<100: # Checking age from 21 to 99
   print("You are eligible")
elif age<=20: # we are checking age from 1 to 20 includes 20 also
   print("You are not eligible")
elif age==0: # we are checking age from 1 to 20 includes 20 also
   print("Newborn")
else:
   print("You are not part of it")

if age <= 12:
    print("Child.")
elif age <= 19:
    print("Teenager.")
elif age <= 35:
    print("Young adult.")
else:
    print("Adult.")
```

Nested if...else Statements

Nested if...else means an if-else statement inside another if statement. We can use nested if statements to check conditions within conditions.

```
age = 70
entry_pass = True

if age >= 60:
    if is_member:
        print("30% discount on all games!")
    else:
        print("20% discount on all games.")
else:
    print("Not eligible for discount.")
```

Match-Case Statement

Match-case statement is Python's version of a switch-case found in other languages. It allows us to match a variable's value against a set of patterns.

```
card_no=int(input("please enter number from 1 to 4"))
match card_no:
    case 1:
        print("Chess")

    case 2:
        print("Carrom")

    case 3:
        print("Basketball")

    case 4:
        print("Volleyball")
```

Ternary Conditional Statement in Python

A ternary conditional statement is a compact way to write an if-else condition in a single line. It’s sometimes called a "conditional expression."

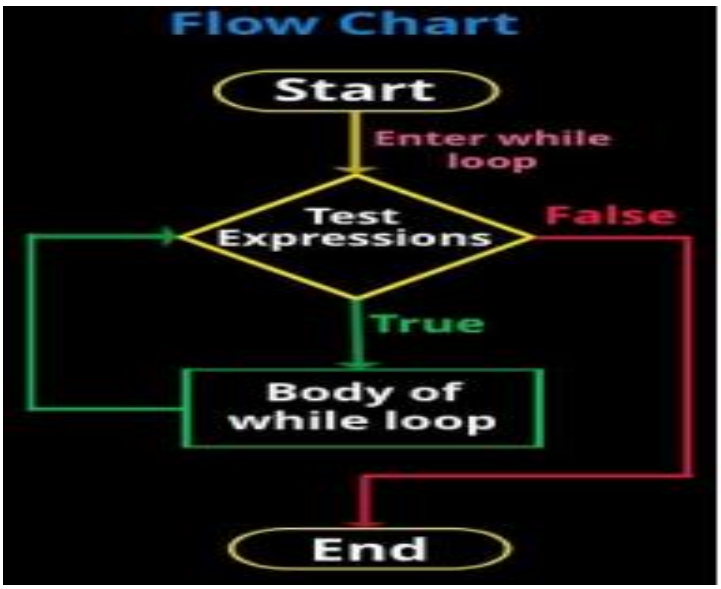
```python
age = 20
s = "Adult" if age >= 18 else "Minor" # First give True condition
# if condition age>18 is true Adult is stored into s var
# if age>18 is false else part execute Minor is stored into s var
print(s)
print("Adult" if age >= 18 else "Minor" ) # this is also same instead of storing into variable s, printing directly
```

Iterators/Loops

Loops are used to repeatedly execute a block of code as long as a specific condition is met or for a predefined number of times. Python supports two main types of loops:

While loop: In while loop as long as the condition is satisfied the set of statements repeats its execution. And once the condition is false the control exits the loop and executes the rest of the code if any.

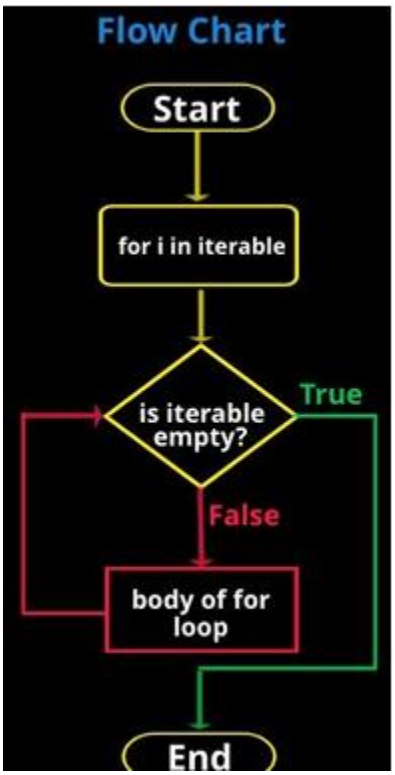
Syntax: while :
body of the loop




```
# Program to print number from 1 to 100
i=1 # this will execute only 1 time at the time of initialization.
# line no. 22 to 24 process goes till the condition is false
# if condition is false then immidiatally it will terminate loop

# here value is 1
while i<=100:  # l
ine 22# initially i=0 exit condition of loop
    print(i) # line 23
    i=i+1 # line 24 # i value is incremented by 1 everytime
# line 23 to 24 (above 2 lines) called body of loop
```

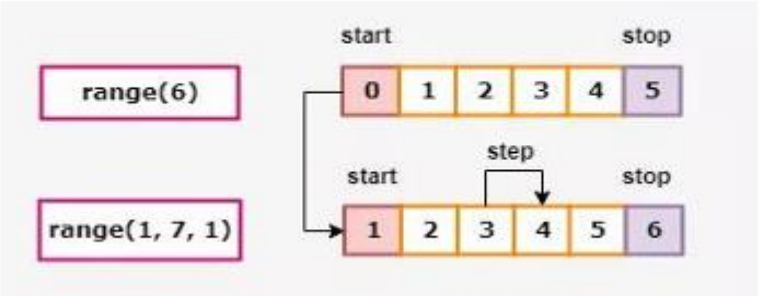
For loop: These are used for sequential traversal. A for loop is used for iterating over a sequence For example: traversing a list or string or array etc.

Syntax: for var in iterable:
body of the loop
[Note: Iterable is something which can work on/with for loop.]



range( ) function : If you want to write for loop in traditional way, that is also possible in python. Where you can specify the number of times your loop should execute by using the function called as range().

Syntax: range(start, stop, step)
Start: Which position to start from. By default it is 0. //optional
Stop: Which position to stop at (not included). //required
Step: Number specifying incrementation. Default is 1. //optional

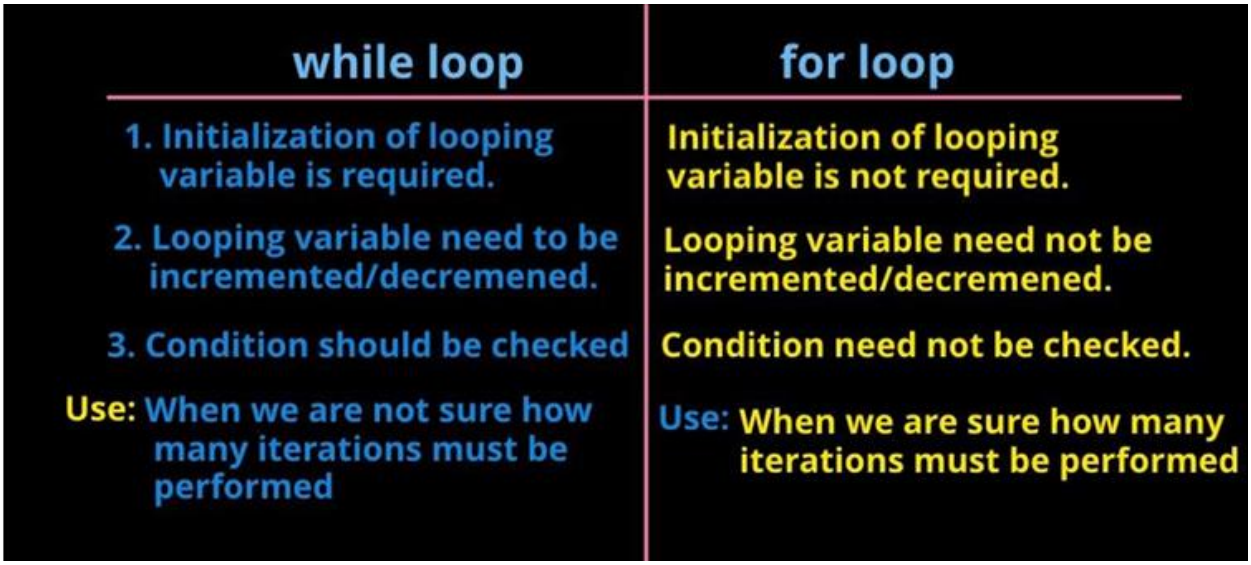



```
# Program to print square of numbers

for i in range(1, 10):
        print(i**2)

# Using range() with For Loop
# The range() function is commonly used with for loops to generate a sequence of
numbers.
# It can take one, two, or three arguments:

# range(stop): Generates numbers from 0 to stop-1.
# range(start, stop): Generates numbers from start to stop-1.
# range(start, stop, step): Generates numbers from start to stop-1, incrementing
by step.

# i=1 stop=10 printing till 10-1=9 steps- bydefault- step is 1, if we give step=2
(it will increment by 2)

for i in range(100):
    print(i) # printing value from 1 to 99
    # we have given only stop value, start value is 1 by default, step is 1 by
default
```

When to use for and while loop ?

While loop must be used when you are not sure how many times a loop should repeat itself. As long as the condition is true the loop must execute.

For loop should be used when you are sure about the number of times the iterations should be performed. Here there is no condition checked.



while loop	for loop
1. Initialization of looping variable is required.	Initialization of looping variable is not required.
2. Looping variable need to be incremented/decremened.	Looping variable need not be incremented/decremened.
3. Condition should be checked	Condition need not be checked.
Use: When we are not sure how many iterations must be performed	Use: When we are sure how many iterations must be performed

Loop Control Statements

Break statement is used to exit the loop prematurely when a certain condition is met. let us see where and why we should use it. Let us consider an example of prime number. A prime number is a number which is divisible by 1 and the number itself. If a number is divisible by any other number 1 and number itself then the number is not a prime number. Break statement can be used in both loops: while and for loop.

```
n=int(input("Enter a number\n"))

for i in range(2,n+1):
    if n%i==0:
        pass
if i==n:
    print(n,"is prime")
else:
    print(n,"is not prime")
```

[Above example will not produce correct output for checking prime number.]

pass is a null statement. The interpreter does not ignore a pass statement, but nothing happens and the statement results into no operation. The pass statement is useful when you don't write the implementation of a function but you want to implement it in the future.

Let us see the above example using break statement

```
n=int(input("Enter a number\n"))

for i in range(2,n+1):
    if n%i==0:
        break
if i==n:
    print(n,"is prime")
else:
    print(n,"is not prime")
```

Explanation of above example

Let us take n=3
i value starts from 2 to 3, range(start, stop)->range(2,3+1)-> loop iterates from 2 to stop-1=3
checking 3%2 ==0(remainder is 1 means false condition), now go to next iteration i value becomes 3
checking 3%3==0 (remainder is 0 means true condition, break statement – it will break the execution of the loop and come out of looping body, i value is 3

If i==n(n value is 3 and i value is also 3)-> condition is true -> means it is prime number

```
#Using break to exit the loop

for i in range(10):
    if i == 5:
        break
    print(i)
```

Explanation: The loop prints numbers from 0 to 9.
When i equals 5, the break statement exits the loop. Execution of loop stops on that line.

continue Statement

The continue statement skips the current iteration and proceeds to the next iteration 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.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Explanation: The loop prints odd numbers from 0 to 9.
When i is even, the continue statement skips the current iteration.

When to Use the else Block

1. **Checking a condition after a loop completes:** Use the else block to perform actions when the loop completes all iterations without interruption.
2. **Searching or matching:** For example, when searching for an element in a list, you can use else to handle cases where the element is not found.
3. **Avoiding redundant flags:** Instead of using a flag variable to check if a loop executed fully, the else block can simplify your code.

```
for num in range(5):
    if num == 3:
        print("Found 3!")
        break
else:
    print("Loop completed without finding 3.")
# If 3 is found, the loop ends with break, and the else block does not execute.
# If 3 is not found, the else block executes.
```

```
count = 0
while (count < 3):
    count = count + 1
```

```
    if count == 2:
        break
    print("while loop body")
print("Outside loop") # this will always execute whether loop breaks or not
```