Boolean Expression

A **Boolean Expression** is an expression that evaluates to either True or False. Comparison/relational operators can be used to create a Boolean expression.
Given below are examples of Boolean expressions :
x == y
x<y
x>=y

Compound Boolean Expressions
Simple Boolean expressions which use comparison operators are combined with logical operators to form compound Boolean expressions. Examples are :
x==y and x==z
x==y or x==z
not a==7 and a<10

Short-Circuit(Lazy) Evaluation
When the second operand in an expression is not evaluated because the result is already known by the first operand is called **short-circuit(lazy) evaluation**.
To make the evaluation fast, Python uses Short-Circuit Evaluation as given below :

X or Y ---> When X is True, don't evaluate Y, result is True
When X is False, evaluate Y, result is what Y returns

X and Y ---> When X is True, evaluate Y, result is what Y returns
When X is False, don't evaluate Y, result is False

Example :
a = 10
b = 20
res = (a==10 or b==30)
Here, no need to evaluate b==30 since a==10 is True and therefore, it doesn't matter if second expression is True or False, the result will be True.

Boolean expressions are used to denote conditions for selection and iterative control statements.

CONTROL STATEMENTS

Control statement determines the flow of the program which is the order in which a program code is executed.

- **Sequential Control** is a form of control in which program code is executed the same order in which code is written line by line.
- **Selective Control** is the form of control that selectively executes instructions in a program code based on some condition.
- **Iterative Control** repeatedly executes instructions and hence control the flow of program based on some condition.

1) **Selection Control**

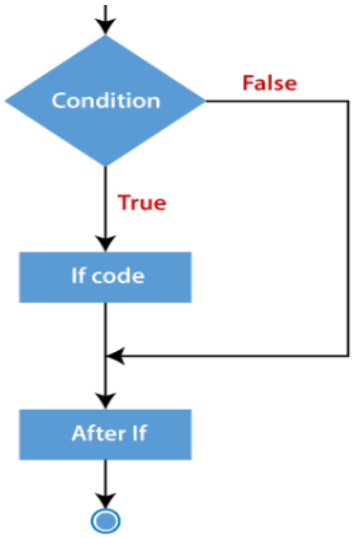
Selection control statements / Decision making statements/ Conditional Statements are available to selectively execute any block of code based on some condition.

Some selection control statements are given below :

(1) **if statement**

Syntax :

```
if condition:
    statement
    statement
    ...
```



If the condition is True, then only the body of **if** is executed.

Example1 :

```
a = 2
if a == 2:
   print("a is 2")
```

Output :

```
a is 2
```

Since the value of the variable **a** is 2, the condition a == 2 becomes True and the statement body of **if** gets executed. Therefore, ''a is 2'' is printed.

Example2 :

```
print("Enter your name : ")
name = input()
if(name=="Adam"):
   print("Your name is Adam")
   print("Welcome")
```

Output :

```
Enter your name : Adam
Your name is Adam
Welcome
```

Here two statements are inside **if**. They both belong to the same block because their **indentation** is same, that is, they are written starting from the same number of spaces from left margin. Therefore, when **if** condition becomes True, both the statements followed by **if** statement are printed.

Example3 :

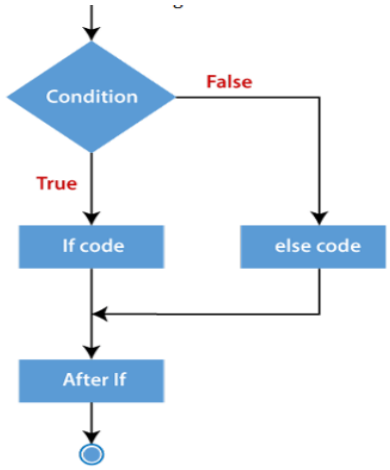
```
age = int(input("Enter age"))
if age >= 18:
     print("Your age is 18+")
     print("You are eligible to vote")
print("Thank you for coming")
```

Here the last statement doesn't belong to **if statement**. Therefore it is a normal statement and is executed even if the condition is True or False.

(2) **if...else Statement**

Syntax :

```
if condition:
    statement
    statement
    ...
else:
    statement
    statement
    ...
```



If the **condition** is True, then the body of if is executed, otherwise the body of else is executed.

Example :

```
a = 14
if a%2 == 0:
   print("Number is even")
else:
   print("Number is odd")
```

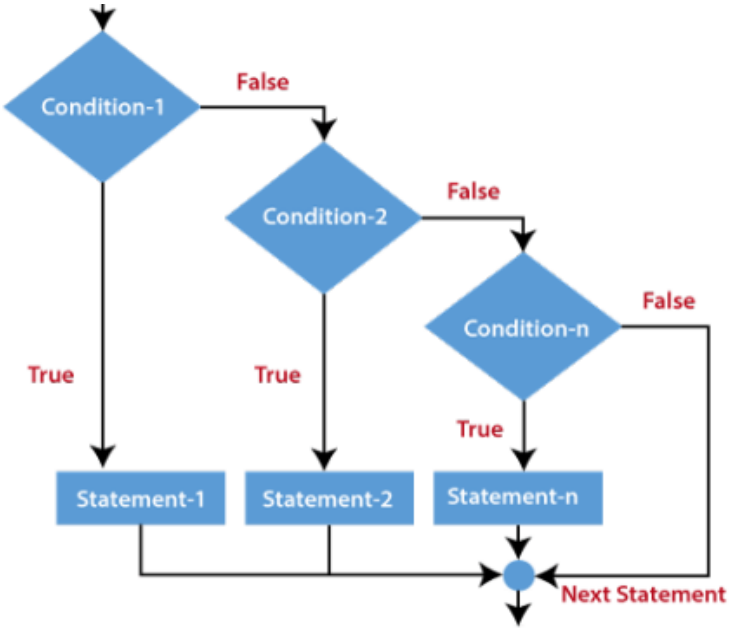
Output :

Number is even

(3) **if...elif...else Statement**

Syntax :

```
if condition:
    statement
    statement
    ...
elif condition:
    statement
    statement
    ...
elif condition:
    statement
    statement
    ...
....
....
else:
    statement
    statement
    ...
```



Firstly, as usual the condition of *if* is checked. If it is True, then the body of *if* is executed.If the condition of *if* is False, then the condition of the first *elif* is checked. If the condition of the first *elif* is True then the statements inside its body are executed, otherwise the next *elif* condition is checked.
If the conditions of *if* and all *elif* are False, then the body of *else* is executed.

Example is given below :

```
a = int(input("Enter a number : "))
if a%2 == 0 and a > 10:
   print("Your number is even and greater than 10")
elif a%2 == 0 and a < 10:
   print("Your number is even and smaller than 10")
else:
   print("Your number is odd")
```

Output:

Enter a number : 8
Your number is even and smaller than 10

(4) **Nested if Statements**

We can use *if*, *if...else* or *if...elif...else* statements in the body of *if*, *elif* or *else*. This is also called **nesting**.

Example :

```
a = int(input("Enter a number"))
if a%2 == 0:
   if a > 10:
      print("Your number is even and greater than 10")
   else:
      print("Your number is even and smaller than or equal to 10")
else:
   print("Your number is odd")
```

Here, we are checking if the number entered by the user is even or odd. If the number is even, then we are further checking if it is greater than 10 or not(**with nested if and else**). Accordingly, we are displaying the message.

pass Statement

The **pass** statement is used to skip from *if*, *elif* or *else*. If we don't want to give anything in the body of if, elif or else, then we give the pass statement in the body. On executing pass, nothing happens.

Example:

```
age = int(input("Enter your age."))
if age < 13:
   print("Hey! kid")
elif age>13 and age < 20:
   pass
else:
   print("You are grown up.")
```

The body of *elif* contains pass. This means that if the age entered by the user is between 13 (included) and 20, then nothing will get executed. Therefore, on giving the age as 15, using pass just skipped and printed nothing.

It is used when we want to do nothing for some specific conditions, like in the last example. Another reason can be that we want to write some code in future but can't leave the body empty as that would throw an error, so we use **pass** statement there.

Iterative Control

Iteration is the execution of same set of code repeatedly. Programming structure that implements iteration is called **loop**.

The two types of iteration are :
1) **Definite Iteration** – Number of times the code block will be executed is specified explicitly at the time loop starts.
2) **Indefinite Iteration** – Number of times the code block will be executed is not specified explicitly in advance.

The **iterative control statements** are the statements that control the repeated execution of a set of code. The two Iterative Control Statements in Python are :
1) **while**
2) **for**

The while Statement

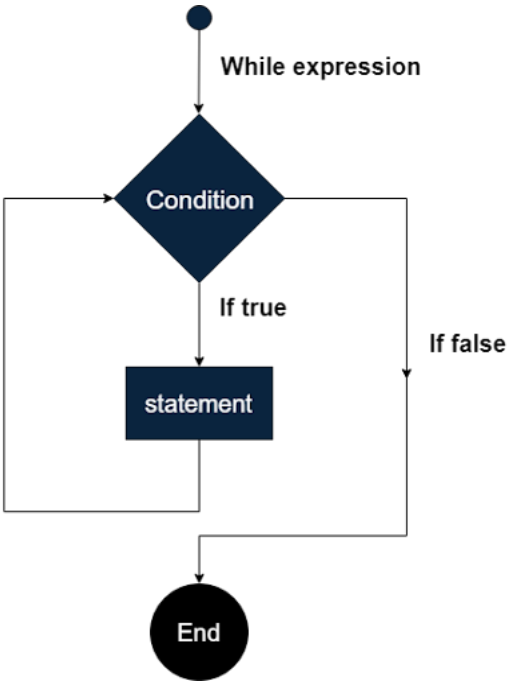
Syntax :

```
while condition:
    statement
    statement
    ...
```

The **body of the while loop** consists of all the ***indented*** statements below **while condition:** .
while loop checks whether the **condition** is True or not. If the condition is True, the statements written in the body of the while loop are executed. Then **again, the condition is checked**, and if found True again, the statements in the body of the while loop are executed again. This process continues until the condition becomes False.

Example :

```
n = 1
while n <= 10:
    print(n)
    n = n + 1
```



OUTPUT

1
2
3
4
5
6
7
8
9
10

The above given code is an example of **definite loop** since we know the loop is going to be executed 10 times before starting the loop.

The code given below is an example of **indefinite loop** because we don't know how many times the loop repeats in advance and the number depends on whether the user inputs 'yes' or not.

```
sel='yes'
while sel=='yes':
      a=int(input("enter value for a : "))
      b=int(input("enter value for b : "))
      print("Sum : ", a+b)
      sel=input("Do you want to continue? ")
```

OUTPUT

```
enter value for a : 4
enter value for b : 5
Sum :  9
Do you want to continue? yes
enter value for a : 1
enter value for b : 2
Sum :  3
Do you want to continue? no
```

Nesting of Loop

Nesting means having one loop inside another loop, i.e., to have a loop inside the body of another loop.

Example :

```
a=1
while a<4:
   b=1
   while b<5:
      print(b,end=' ')
      b=b+1
   print("\n")
   a=a+1
```

OUTPUT

```
1 2 3 4

1 2 3 4

1 2 3 4
```

Infinite Loop

An infinite loop will keep on executing its body forever. Below is an infinite loop created using a while loop.

```
while True:
   print(1)
```

The above code will continue to display 1 infinitely. Press ctrl+c to stop infinite loops.

The 'for' statement

Syntax :

```
for variable in sequence:
   statement
   statement
   ...
```

sequence is a sequence like list, tuple, string, etc. **variable** is a variable that takes the value of each item in the sequence.

Example :

```
colors = ["Blue", "Green", "Yellow", "Orange", "Red", "Brown"]
for c in colors:
   print(c)
```

OUTPUT

```
Blue
Green
Yellow
Orange
Red
Brown
```

colors is a list here containing six items.
c is a variable which goes to each element in the list **colors** and takes its value.
So, in the first iteration, **c** is equal to the 1st element of the list. In the second iteration, **c** is equal to the 2nd element of the list, and so on.

range() function

The range() function is used to generate a sequence of numbers.

Syntax :

```
range(start, stop, step_size)
```

start - Integer from which the sequence starts. It is optional. Its default value is 0.
stop - Integer before which the sequence stops. It is mandatory.
step_size - Increment between each integer in the sequence. It is optional. Its default value is 1.

Example :

```
range(10) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(1, 10) = [1, 2, 3, 4, 5, 6, 7, 8, 9]
range(1, 10, 2) = [1, 3, 5, 7, 9]
```

range() function can be used with for loop to generate a sequence and iterate through it.

Example :

```
for n in range(1, 5):
    print(n)
```

OUTPUT

```
1
2
3
4
```

'break' and 'continue' statements

In Python, we can jump out of a loop or jump to the starting condition of a loop whenever we want. We do this with the help of **break** and **continue** statements respectively.

break statement

break is used to break or terminate a loop whenever we want. Just type **break** after the statement after which you want to break the loop.

Example :

```
for n in range(1, 6):
    if n == 3:
        break
    print(n)
```

OUTPUT

```
1
2
```

continue statement

continue statement works similar to the break statement. The only difference is that **break** statement terminates the loop whereas **continue** statement skips the rest of the statements in the loop and starts the next iteration.

Example :

```
for n in range(1, 6):
    if n == 3:
        continue
    print(n)
```

OUTPUT

1
2
4
5

Nested loop

Nested loop is the one in which a loop is inside another loop. In the code given below, **while** loop is nested inside **for** loop.

Example :

```
for a in range(1,5,1):
    b=1
    while b<3:
        print("b is : ",b)
        b=b+1
```

OUTPUT

b is : 1
b is : 2
b is : 1
b is : 2
b is : 1
b is : 2
b is : 1
b is : 2