Python Loop

- for Loop
- while loop
- nested loops

for loop

for loops can iterate over a sequence of iterable objects in python. Iterating over a sequence is nothing but iterating over strings, lists, tuples, sets and dictionaries.

In [1]:
```
# list of sequence to print each value using for loop

seq = [1,2,3,4,5]

for i in seq:
    print(i)
```

```
1
2
3
4
5
```

In [2]:
```
# list of sequence to print statement using for loop

for i in seq:
    print('Yep')
```

```
Yep
Yep
Yep
Yep
Yep
```

In [3]:
```
# list of sequence to print self addition of each value using for loop

for i in seq:
    print(i+i)
```

```
2
4
6
8
10
```

iterating over a string

In [4]:
```python
name = 'Welcome to punjab'
for i in name:
    print(i, end="#")
```

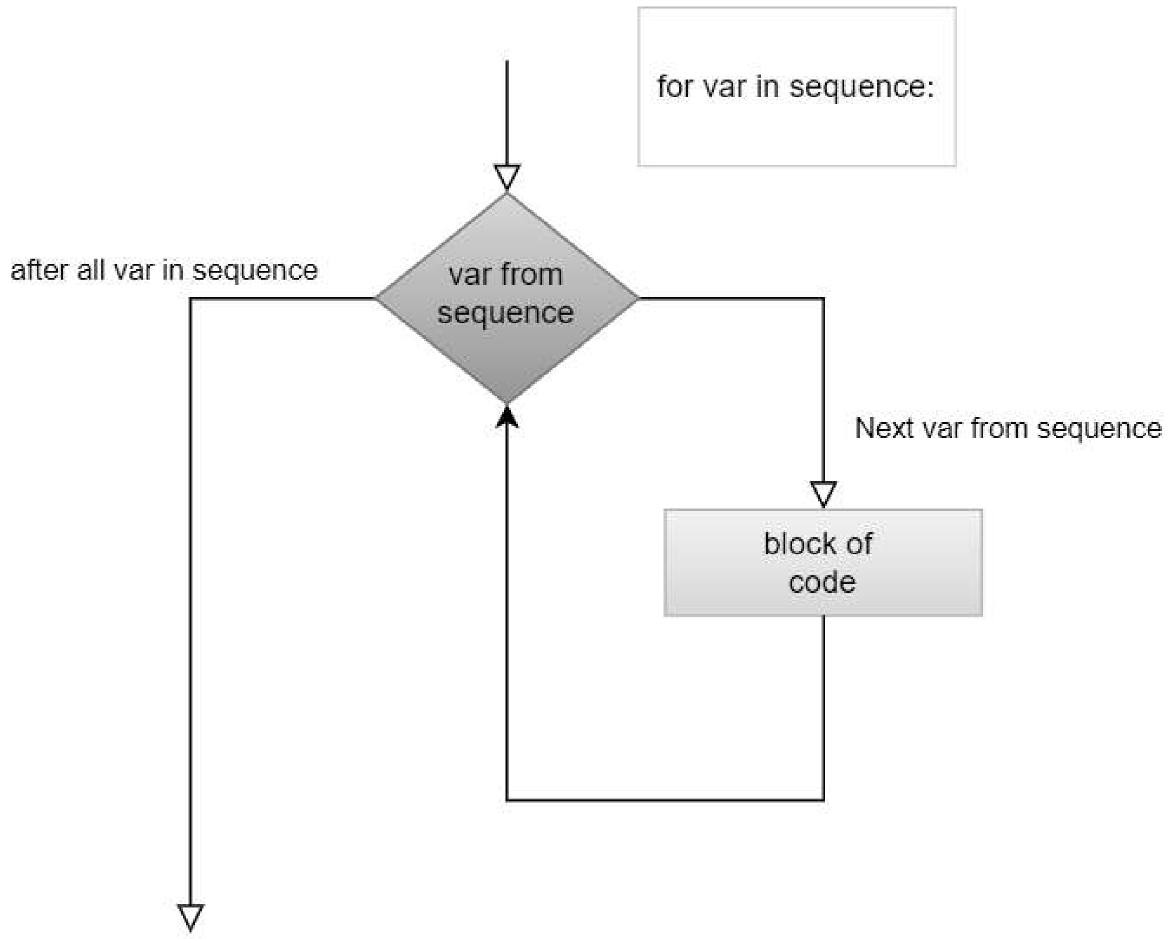
```
W#e#l#c#o#m#e# #t#o# #p#u#n#j#a#b#
```

Iterating over a tuple

In [5]:
```python
colors = ("Red", "Green", "Blue", "Yellow")
for x in colors:
    print(x)
```

```
Red
Green
Blue
Yellow
```

Similarly, we can use loops for lists, sets and dictionaries.



What if we do not want to iterate over a sequence? What if we want to use for loop for a specific number of times?

Here, we can use the range() function.

```
In [6]: for k in range(5):
            print(k)
```

```
0
1
2
3
4
```

we can also loop over a specific range

```
In [7]: for k in range(3,8):
            print(k)
```

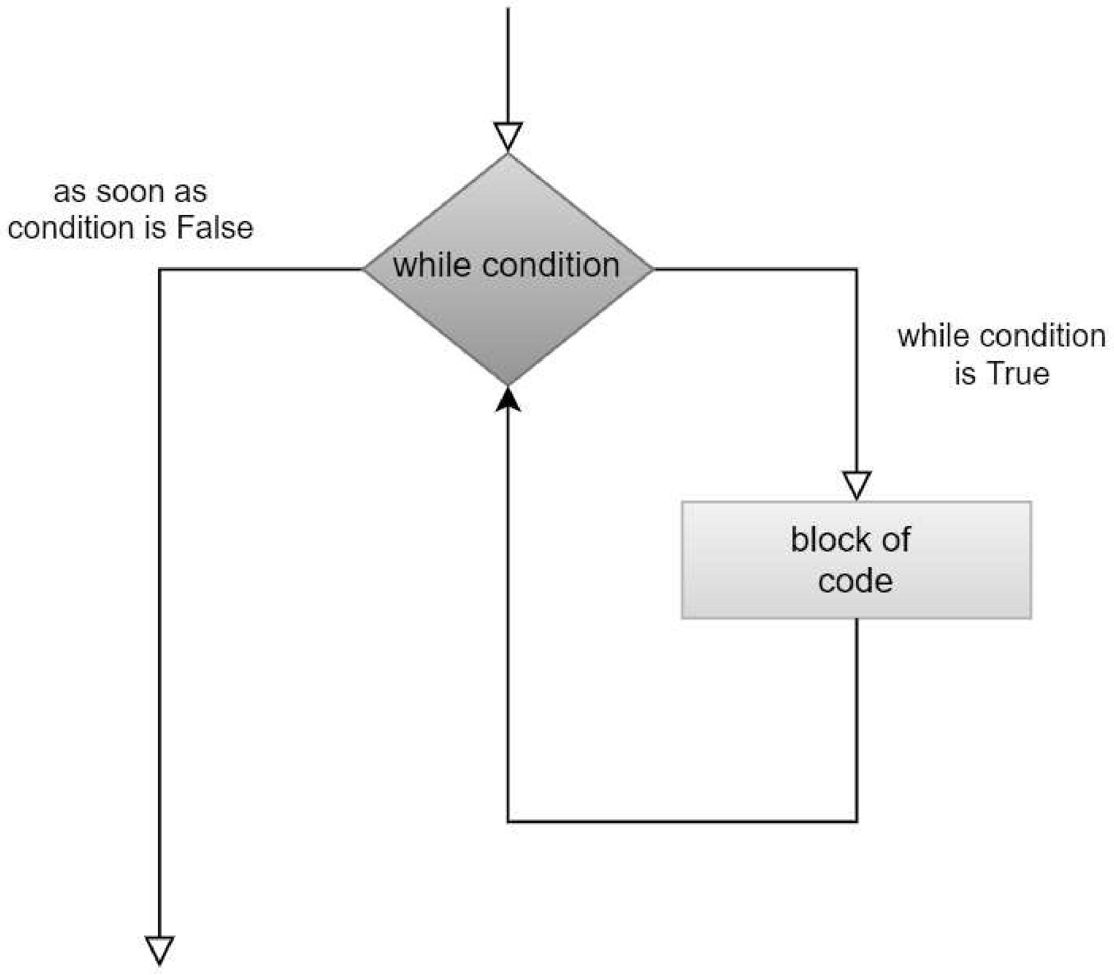
```
3
4
5
6
7
```

```
In [8]: m = list(range(5))
        m
```

```
Out[8]: [0, 1, 2, 3, 4]
```

while Loop

while loops execute statements while the condition is True. As soon as the condition becomes False, the interpreter comes out of the while loop.



In [9]:
```
count = 5
while (count > 0):
    print(count)
    count = count - 1
```

```
5
4
3
2
1
```

We can also use else statement with the while loop. it runs when the while loop condition becomes False, the interpreter comes out from the while loop and the else statement is executed.

In [10]:
```
x = 5
while (x > 0):
    print(x)
    x = x - 1
else:
    print('counter is 0')
```

```
5
4
3
2
1
counter is 0
```

In [11]:
```
i = 1
while i < 5:
    print('i is: {}'.format(i))
    i = i+1
```

```
i is: 1
i is: 2
i is: 3
i is: 4
```

Nested Loops

use loops inside other loops, such types of loops are called as nested loops

In [12]:
```
# nesting for loop in while loop
i=1
while (i<=3):
    for k in range(1, 4):
        print(i*k)
    i = i + 1
    print()
```

```
1
2
3

2
4
6

3
6
9
```

In [13]:
```
# nesting while loop in for loop

for i in range(1, 4):
    k = 1
    while (k<=3):
        print(i*k)
        k = k + 1
    print()
```

```
1
2
3

2
4
6

3
6
9
```

Control Statements

Use with for and while loops to change their behaviour.

They are pass, continue and break.

pass

In [14]:
```
i=1
if (i == 2):
```

```
  Input In [14]
    if (i == 2):
                ^
IndentationError: expected an indented block
```

To avoid such an error and to continue the code execution, pass statement is used. pass statement acts as a placeholder for future code.

In [15]:
```
i=1
if (i == 2):
    pass
```

```
In [16]: for j in range(5):
             pass
```

continue

This keyword is used in loops to end the current iteration and continue the next iteration of the loop

```
In [17]: for i in range(1,10):
             if(i%2 == 0):
                 continue
             print(i)
```

```
1
3
5
7
9
```

```
In [18]: i = 1
         while (i <= 10):
             i = i + 1
             if (i%2 != 0):
                 continue
             print(i)
```

```
2
4
6
8
10
```

break

break keyword is used to bring the interpreter out of the loop and into the main body of the program

```
In [19]: i = 1
         while (i <= 10):
             i = i + 1
             if (i == 5):
                 break
             print(i)
```

```
2
3
4
```

In [20]:
```
for i in range(1, 10):
    if (i == 5):
        break
    print(i)
```

```
1
2
3
4
```

Break: eliminates the execution of remaining iteration of loop

Continue: terminate only the current iteration of loop.

list comprehension

List comprehensions are used for creating new lists from other iterables like lists, tuples, dictionaries, sets, and even in arrays and strings.

Syntax:

List = [expression(item) for item in iterable if condition]

expression: it is the item which is being iterated.

iterable: it can be list, tuples, dictionaries, sets, and even in arrays and strings.

condition: condition checks if the item should be added to the new list or not.

In [21]:
```
x = [1,2,3,4]
```

In [22]:
```
# square the value present in list

m = [i**2 for i in x]
m
```

Out[22]:
```
[1, 4, 9, 16]
```

In [23]:
```
for i in x:
    print(i**2)
```

```
1
4
9
16
```

```
In [24]: # accepts items with the small letter “a” in the new list

         names = ["Kartik", "Jyoti", "Anchal", "Vishu", "Piyush"]
         m = [i for i in names if "a" in i]
         print(m)
```

```
['Kartik', 'Anchal']
```

Frozen set

```
In [25]: mylist = ['apple', 'banana', 'cherry']
         x = frozenset(mylist)
         print(x)
```

```
frozenset({'banana', 'apple', 'cherry'})
```

```
In [26]: # Initializing a dictionary
         person = {"name": "Himani", "age": 21, "Country": "India"}

         # Initializing a frozenset object using a dictionary
         frozenDict = frozenset(person)

         # Assigning an item to the frozenset object
         frozenDict[1] = "Work"

         # Printing the frozenset object
         print('The frozenset object:', frozenDict)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [26], in <cell line: 8>()
      5 frozenDict = frozenset(person)
      7 # Assigning an item to the frozenset object
----> 8 frozenDict[1] = "Work"
     10 # Printing the frozenset object
     11 print('The frozenset object:', frozenDict)

TypeError: 'frozenset' object does not support item assignment
```

Thank you