While Loop

- A while loop in Python is used for repeating a block of code as long as a specified condition remains True. The loop checks the condition before executing the code block each time.

In [1]:
```
i=1 #intializing
while i<=5: #condition
    print('loop entered',i,'Time') #indentation
    i=i+1 #increment to avoid infinite loop
```

```
loop entered 1 Time
loop entered 2 Time
loop entered 3 Time
loop entered 4 Time
loop entered 5 Time
```

In [2]:
```
i=5
while i>=1:
    print('loop entered',i,'Time')
    i=i-1 #decrement
```

```
loop entered 5 Time
loop entered 4 Time
loop entered 3 Time
loop entered 2 Time
loop entered 1 Time
```

Nested While

- A nested while loop in Python is when we place one while loop inside another. This allows us to iterate over multiple levels of conditions.
- The inner loop completes all iterations before the outer loop increments. This structure is often used in working with multi-dimensional data or nested processes.

In [3]:
```
i=1
while i<=3:
    print('OuterLoop',i)
    j=1
    while j<=2:
        print('InnerLoop',j)
        j=j+1
    i=i+1
    print()
```

```
OuterLoop 1
InnerLoop 1
InnerLoop 2

OuterLoop 2
InnerLoop 1
InnerLoop 2

OuterLoop 3
InnerLoop 1
InnerLoop 2
```

In [4]:
```
i=1
while i<=3:
    print('OuterLoop',i,end='    ') #if we mention end the new line will not crea
    j=1
    while j<=2:
        print('InnerLoop',j,end='    ')
        j=j+1
    i=i+1
    print()
```

```
OuterLoop 1    InnerLoop 1    InnerLoop 2
OuterLoop 2    InnerLoop 1    InnerLoop 2
OuterLoop 3    InnerLoop 1    InnerLoop 2
```

In [5]:
```
i=1
while i<=4:
    j=1
    while j<=5:
        print(i*j,end='    ')
        j+=1
    i+=1
    print()
```

```
1    2    3    4    5
2    4    6    8    10
3    6    9    12    15
4    8    12    16    20
```

For Loop

- A for loop in Python is used for iterating over a sequence (like a list, tuple, dictionary, string, or range). Unlike the while loop, which depends on a condition, a for loop loops through items in a collection.

In [6]:
```
name='Tharun'
for i in name:
    print(i)
```

```
T
h
a
r
u
n
```

In [7]:
```
fruits=['apple','banana','grapes','oranges']
for i in fruits:
    print('I Like',i)
```

```
I Like apple
I Like banana
I Like grapes
I Like oranges
```

In [8]:
```
range(5)
```

Out[8]:
```
range(0, 5)
```

In [9]:
```
for i in range(5):
    print(i,end=' ')
```

```
0 1 2 3 4
```

In [10]:
```
for i in range(00,51):
    if(i%5==0):
        print(i,end='  ')
```

```
0  5  10  15  20  25  30  35  40  45  50
```

Break

- In Python, the break statement is used to exit a loop before it naturally completes. It’s commonly used when a specific condition is met.

In [11]:
```
i = 1
while i <= 5:
    print('Loop entered', i, 'Time')
    if i == 3:  # Break when i equals 3
        print('Breaking the loop...')
        break
    i += 1
```

```
Loop entered 1 Time
Loop entered 2 Time
Loop entered 3 Time
Breaking the loop...
```

In [12]:
```
for num in range(1, 6):
    if num == 4:
        print('Breaking at', num)
        break
    print('Number:', num)
```

```
Number: 1
Number: 2
Number: 3
Breaking at 4
```

Continue

- The continue statement in Python is used to skip the current iteration of a loop and jump to the next one without breaking out of the loop completely.

In [13]:
```
for num in range(1, 6):
    if num == 3:  # Skips when num equals 3
        print('Skipping:', num)
        continue
    print('Number:', num)
```

```
Number: 1
Number: 2
Skipping: 3
Number: 4
Number: 5
```

In [14]:
```
i = 0
while i < 5:
    i+=1
    if i == 3:  # Skips when i is 3
        print('Skipping', i)
        continue
    print('Loop entered', i, 'Time')
```

```
Loop entered 1 Time
Loop entered 2 Time
Skipping 3
Loop entered 4 Time
Loop entered 5 Time
```

Pass

- The pass statement in Python acts as a placeholder, allowing code to run without executing anything within a block.

In [15]:
```
for num in range(1, 6):
    if num == 3:
        pass  # Does nothing but keeps the structure intact
    else:
        print('Number:', num)
```

```
Number: 1
Number: 2
Number: 4
Number: 5
```

For-Else

- It is not supported in other languages
- In Python, the for...else statement is a unique construct where the else block executes only if the for loop completes all iterations without encountering a break statement.

In [16]:
```
numbers = [1, 2, 3, 4, 5]

for num in numbers:
    if num == 3:
        print('Found 3!')
        break  # Exiting the loop before completing
```

```
else:
    print('Loop completed without a break.')
```

```
Found 3!
```

In [17]:
```
for i in range(1, 6):
    print('Checking:', i)
    if i == 10:  # Condition never met
        break
else:
    print('Loop completed without finding 10.')
```

```
Checking: 1
Checking: 2
Checking: 3
Checking: 4
Checking: 5
Loop completed without finding 10.
```

In [18]:
```
for i in range(1, 6):
    print('Checking:', i)
    if i==2:
        break
else:
    print('Loop completed without finding 10.')
```

```
Checking: 1
Checking: 2
```

In [19]:
```
num=int(input('Enter a number to Check Prime or Not:'))
for i in range(2,num):
    if(num%i==0):
        print('Not an Prime Number')
        break
else:
    print('Prime Number')
```

```
Prime Number
```

In [20]:
```
num=int(input('Enter a number to Check Prime or Not:'))
for i in range(2,num):
    if(num%i==0):
        print('Not an Prime Number')
        break
else:
    print('Prime Number')
```

```
Not an Prime Number
```