

Section-2: Repetition Structures: Introduction, while loop, for loop, Calculating a Running Total, Input Validation Loops, Nested Loops.

Repetition or Iterative or Looping Structures/Statements

Introduction

In Python, **repetition structures** are used to execute a block of code repeatedly. These structures allow you to execute the same code repeatedly for a finite number of times or until a condition is satisfied.

Each repetition of a block of code is known as a loop or an iteration. So, the repetition structures are also called **looping or iterative structures**.

There are two types of repetition structures in Python..

1. **Indefinite or Condition Controlled Loop** - A loop that repeats an action until the program finds that it needs to stop based on a condition. (**while loop**)
2. **Finite or Sequence Controlled Loop** - A loop that repeats a block of code a predefined number of times or over a sequence of elements. (**for loop**)

Python provides two repetition or looping or iterative statements,

1. **'while' loop**
2. **'for' loop**
3. **Nested loops**

1. **The 'while' loop:**

Definition:

A **"while" loop** executes a block of statements repeatedly until the given condition is **True**.

The "while" loop is used when we DO NOT KNOW the number of iterations.

Entry controlled or a Pre-Test loop because the **'while' loop** first checks the "condition" to decide if it needs to execute the block of statements.

Event-controlled loop because the termination of the **'while' loop** depends on an event instead of executing a fixed number of times.

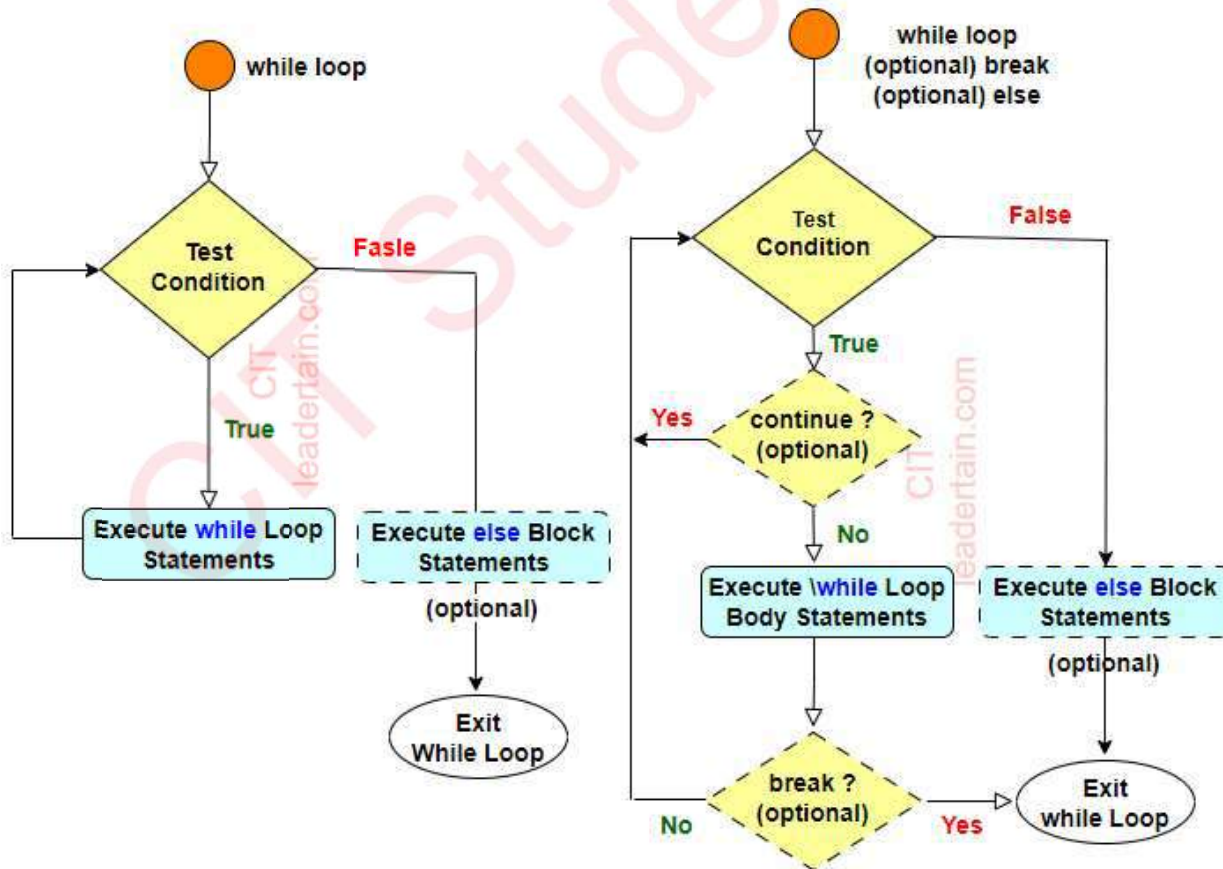
Syntax of 'while' loop:

<pre> Initialization (optional) while (condition): Loop Body statements Incr or Decr (optional) else: (optional) Block of statements </pre>	<pre> Initialization (optional) while (condition): if (condition): continue (optional) if (condition): break (optional) Loop Body statements Incr or Decr (optional) else: (optional) Block of statements </pre>
---	--

Different ways to write 'while' loop condition:

while(true) statements	while(i<5) statements	while(i<=n) statements
----------------------------------	-------------------------------------	--------------------------------------

Flow Chart of 'while' loop



WHILE loop:

- Here, the **condition** is a boolean expression that is evaluated before each iteration of the loop. If the condition is **True**, the code inside the loop is executed. This will continue until the condition becomes **False**.

Application: Write a program to print 1-5 using a 'while' loop.

```
# program to print 1-5 using a 'while' loop.
i = 1
# while loop for i = 1 to 5
while i <= 5:
    print(i)
    i += 1
```

Output:

```
1
2
3
4
5
```

Explanation: The while loop continued as long as i is less than or equal to 5. The i += 1 statement increments the value of i by 1 on each iteration of the loop.

WHILE loop with ELSE:

- When the **'while'** condition becomes **False**, the loop checks for the **optional 'else'** block.
 - If **'else'** block is available, it executes the **'else'** block and then exits the loop.
 - If **'else'** block is not available, then simply exits the loop.

Application: Write a program to print 1-5 using a 'while' loop with 'else'

```
# program to print 1-5 using a 'while' loop and 'else' block
i=1
# while loop for i = 1 to 5
while i <= 5:
    print(i)
    i += 1
else:
    print("Reached end of the loop")
```

Output:

```
1
2
3
```

4

5

Reached end of the loop

Explanation: The while loop continued until from 1 to 5. Once the loop is complete, the 'else' block is executed. Then, exited the loop.

The `i += 1` statement increments the value of `i` by 1 on each iteration of the loop.

Application:

Aim: Generate Fibonacci series up to a given number of terms

```
n = int(input("Enter how many Fibonacci terms : "))
```

```
i = 0
```

```
# Term1 and Term2
```

```
term1, term2 = 0, 1
```

```
# Is the nth term positive?
```

```
if n <= 0:
```

```
    print("Enter a positive integer>0.")
```

```
# If n is only 1 term
```

```
elif n == 1:
```

```
    print("Fibonacci series of",n,"terms is:")
```

```
    print(term1)
```

```
# Find and generate Fibonacci series up to n term
```

```
else:
```

```
    print("Fibonacci series of",n,"terms: ")
```

```
    while i < n:
```

```
        print(term1, end=" ")
```

```
        next = term1 + term2
```

```
        term1 = term2
```

```
        term2 = next
```

```
        i+=1
```

Output:

Enter how many Fibonacci terms: 5

Fibonacci series of 5 terms:

0 1 1 2 3

2. The 'for' loop:

Definition:

- A for loop is used to iterate over a sequence of elements such as string, range(), list, set, tuple or dictionary.
- The code inside the loop is executed repeatedly once for each element in the sequence.
- The "for" loop is used when we KNOW number of iterations.

Syntax: for

```
for var in sequence:
    Loop body statements

else: (optional)
    Block of statements
```

```
for var in sequence:
    if (condition):
        continue (optional)
    if (condition):
        break (optional)
    Loop body statements
else: (optional)
    Block of statements
```

var - an iterator variable

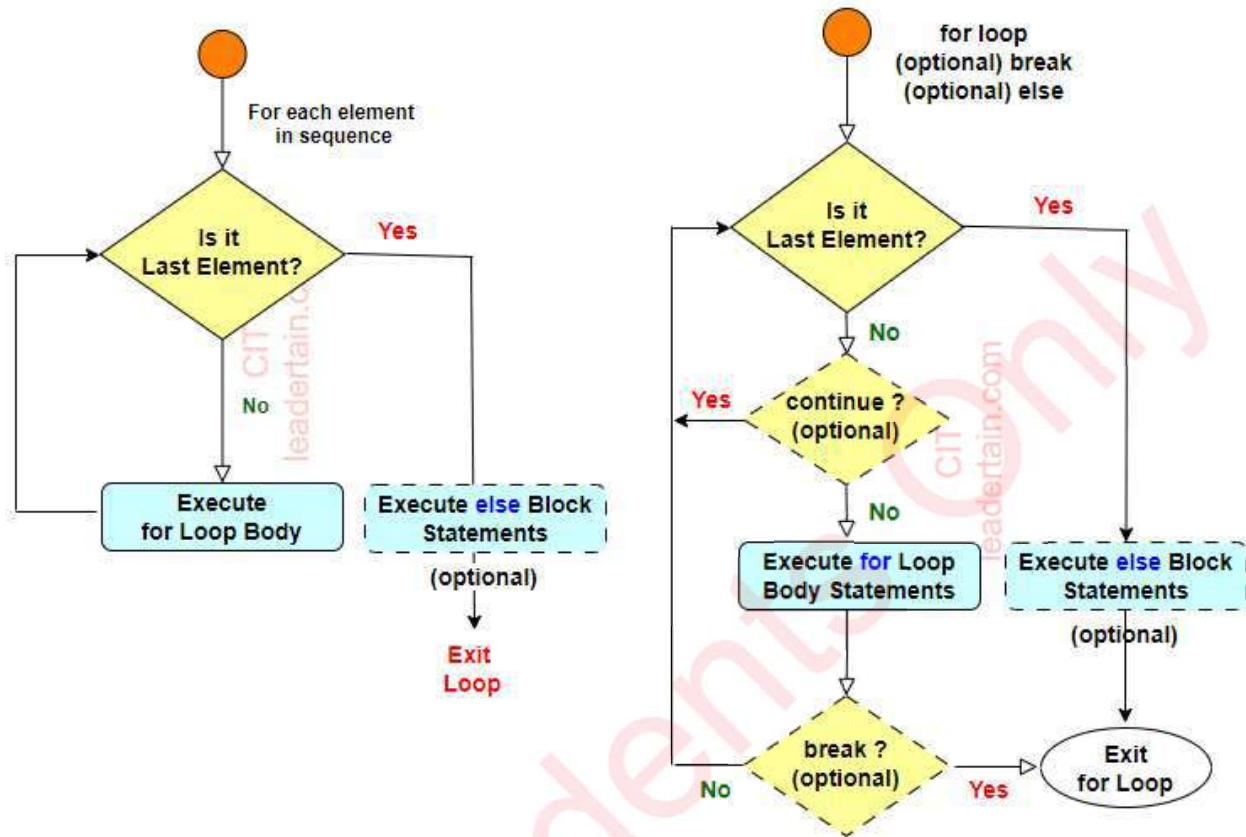
sequence - a sequence of elements; a sequence can be a string, range(), list, set, tuple or dictionary

Loop body statements - a block of for loop statements

else - is an optional block in 'for'. When the for loop completes, it enters 'else' block of statements.

- **var** is an **iterator variable** that takes one element at a time from the **sequence** on each iteration.
- After taking the element in **var**, the **loop statements** execute.
- **for** loop continues until the last value of the sequence is reached.

Flow chart - for loop



Uses of for loop:

for loop through values or a string: iterates through each value or each character of a string sequence

```
# for loop using values sequences
print("Iterate data sequence")
for i in (10,20,30,40,50): # for block is exected for each value
    print(i, end=' ')
else:
    print("\nEnd of the loop")
```

Output:

Iterate data sequence
10 20 30 40 50
End of the loop

```
# for loop using string sequence
print("Iterate string sequence")
for i in "CIT Python": # for block is exected for each character
    print(i, end=' ')
```


Output:

Iterate string sequence
C I T P y t h o n

for loop through a list: iterates through each element of a list/set/tuple/dict sequences

```
# for loop using list sequence
print("Iterate a list")
# for block is exected for each element of the list
branches=["CIT","CSE", "AI", "AIML","ECE"]
for i in branches:
    print(i, end=' ')

```

Output:

Iterate a list
CIT CSE AI AIML ECE

for loop using range() function: iterates through a range of values in sequence

for loop using sequence of range() function

Method-1: range(end value)

Default start value is 0

Parameter - is the end value, (Goes up to end - 1, not including end value)

Default Increment by 1

```
print("Iterate in range(stop)")
for i in range(5): # 0-4, the 5 not included
    print(i)

```

Output:

Iterate in range(stop)

0
1
2
3
4

Method-2: range(start, end value)

Parameter-1 is start value,

Parameter-2 is end value, (Goes up to end - 1, not including end value)

Default - Increment by 1

```
print("Iterate in range(start, stop)")    #
for i in range(1,5): # 1-4, the 5 not included
    print(i)
```

Output:

Iterate in range(start, stop)

1
2
3
4

Method-3: range(start, end, incr/decr value) -

Parameter-1 is start value,

Parameter-2 is end value, (Goes up to end - 1, not including end value)

Parameter-3 is Increment or decrement value.

```
print("Iterate in range(start, stop, inc/dec)")
for i in range(1,5,2): # 1,3 the 5 not included
    print(i)
```

Output:

Iterate in range(start, stop, inc/dec)

1
3

Application-1: Program to count number of even integers in the given list using for loop.

for loop: Program to count the number of even integers in a list.

List of integer numbers

```
numbers = [10, 5, 7, 4, 20, 37, 9]
```

variable to track the even count

```
ecount = 0
```

repete over the list

```
for n in numbers:
```

```
    if n % 2 == 0:
```

```
        ecount += 1
```

```
print("Count of even numbers is", ecount)
```

Output:

Count of even numbers is 3

Application-2: Write a program to Find GCD of 2 numbers

```

# Storing user input into num1 and num2
num1 = int(input("Enter integer number1 : "))
num2 = int(input("Enter integer number2 : "))
# identify smallest of 2 numbers and assign to limit
if(num1<num2):
    limit = num1
else:
    limit = num2
for i in range(1,limit+1):
    # Checks if the current value of i is
    # factor of both the integers num1 & num2
    if(num1%i==0 and num2%i==0):
        gcd = i
print(f"GCD of input numbers {num1} and {num2} is: {gcd}")

```

Output:

```

Enter integer number1 : 50
Enter integer number2 : 100
GCD of input numbers 50 and 100 is: 50

```

Application-3: Write a program to print ASCII value & character set in Python

```

print("ASCII ==> Character\n");
for i in range(0,127):
    print(f"{i} ==> {chr(i)}")

```

Output:

```

...
120 ==> x
121 ==> y
122 ==> z
123 ==> {
124 ==> |
125 ==> }
126 ==> ~

```

Calculating Running Total**Definition:**

A running total is the sum of numbers that accumulates over a sequence of numbers. To calculate a running total in Python, you can use a loop to iterate through a range or list of numbers and keep track of the running total using a variable.

For example, if you have a list of numbers [1, 2, 3, 4], the running total would be [1, 3, 6, 10], where each element is the sum of all the elements that came before it.

Purpose:

A running total provides subtotals for any further calculations or for preparing a report.

Application:

```
'''
Program to find running total within a given range (using for loop)
'''
runningTotal = 0;
num = int(input("Enter +ve integer 1-100 : "))
if(num<0 or num>100):
    print("Out of range")
    exit(0)

# for loop terminates when num is less than count
for i in range(num+1):
    runningTotal += i
    print(f"{i}    {runningTotal} ")
```

Output:

```
Enter +ve integer 1-100 : 5
0 0
1 1
2 3
3 6
4 10
5 15
```

Explanation:

The program takes an input number between 1 and 100 from the user. If the given number is outside the range of 1-100, then the programs exits. If the given number is with in the range of 1-100 then the running total is calculated in 'for' loop and prints the result for each iteration.

Input Validation Loop**Definition:**

An input validation loop prompts the user to enter input data, checks the input for validity, and repeats the prompt until valid input is entered. The loop continues until the user enters valid input and then the program can proceed with the remaining steps.

Purpose:

Input validation loops in Python ensure that the user enters valid input data. This is important because invalid data can cause errors or unexpected behavior in the program.

Application:

```
'''
Aim: Check the input number is valid. If invalid, then repeat the
prompt to reenter another number
'''
while True:
    user_input = input("Enter a number between 1 and 10 : ")
    num = int(user_input)
    if num < 1 or num > 10:
        print("Invalid number.")
    else:
        print("Valid number.")
        break
```

Output:

```
Enter a number between 1 and 10 : 27
Invalid number.
Enter a number between 1 and 10 : 7
Valid number.
```

Explanation:

In this example, the loop continues until the user enters a valid number between 1 and 10. The input is first converted to an integer using the `int()` function. If the input is an invalid integer, the loop continues. If the input is a valid and within the range, the loop is exited and the program can proceed with the remaining steps.

Nested Loops in Python

Definition:

Nested loop in Python is a loop that is placed inside another loop. The nested loops are used to iterate over multiple groups of data or to perform a task repeatedly for each element of multiple lists or collections of data.

We have **for** and **while** loops in Python. We can nest these loops in any combination in Python. Two such combinations are as follows:

- **for** loop nested with another **for** loop,
- **for** loop nested with a **while** loop

Purpose:

Nested loops are typically used for working with patterns, multidimensional data structures, such as printing two-dimensional arrays, iterating a list that contains a nested list.

General Syntax: Nested Loop in Python

OuterLoop Expression:

InnerLoop Expression:
Statements inside InnerLoop

Statements inside Outer_Loop

Syntax: Nested for Loops

```
for outer_var in outer_sequence:
```

```
for inner_var in inner_sequence:
    Statements in inner for loop
```

```
Statements in outer for loop
```

Note: Each iteration of the outer for loop triggers a complete iteration of the inner for loop.

Syntax: Nested for - while Loops

```
for outer_var in outer_sequence:
```

```
while (condition):
    Statements in inner while loop
```

```
Statements in outer for loop
```

Note: Each iteration of the outer for loop triggers a complete iteration of the inner while loop.

Application-1:

Lab-5: Use a for loop to print a triangle using *s.
 # Allow the user to specify how high the triangle should be.

```
rows = int(input('Enter rows: '))
for i in range(1,rows+1):
    for j in range(0,i):
        print('*', end=' ')
    print('')
```

Output:

Enter rows: 5
*
**

Application-2:

#Program to print multiplication tables using nested "for" loop
 # Outer loop - tables 5 and 6

```
for i in range(5, 7):
    # Inner loop from 1 to 10
    for j in range(1, 11):
        print(i, "*", j, "=", i*j)
    print()
```

Output:

5 * 1 = 5	6 * 1 = 6
5 * 2 = 10	6 * 2 = 12
5 * 3 = 15	6 * 3 = 18
5 * 4 = 20	6 * 4 = 24
5 * 5 = 25	6 * 5 = 30
5 * 6 = 30	6 * 6 = 36
5 * 7 = 35	6 * 7 = 42
5 * 8 = 40	6 * 8 = 48
5 * 9 = 45	6 * 9 = 54
5 * 10 = 50	6 * 10 = 60

Application-3:

#Program to print a number pattern using nested "while" loop

```
i=1
while i<=5:
    j=1
    while j<=i:
        print(j,end=" ")
        j=j+1
    print("")
    i=i+1
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Application-4:

#Find Prime numbers in an Interval using nested "for - while" loop

```
lownum = int(input("Enter low number of interval : "))
highnum = int(input("Enter high number of interval : "))
print("Prime numbers between", lownum, "and", highnum, "are:")
for num in range(lownum, highnum + 1):
    # Primes are always > 1
    if num > 1:
        i = 2
        while (i<num):
            if (num % i) == 0:
                break
            i += 1
        else:
            print(num, end=' ')
```

Output:

```
Enter low number of interval : 1
Enter high number of interval : 20
Prime numbers between 1 and 20 are:
2 3 5 7 11 13 17 19
```


Jump Statements in Python

The Jump Statements are loop Control Statements in Python. The loop Control Statements are used to change the normal flow of a loop based on a condition.

The 3 jump or loop control statements in Python are,



1. **break** statement
2. **continue** statement
3. **pass** statement

1. “break” statement

Definition:

- The break statement in Python is used to terminate the loop and brings the control out of the loop. The break statement is used in both the **while** and the **for** loops.
- The break statement is especially useful to quit from a nested loop (loop within a loop). It terminates the inner loop and control shifts to the statement in the outer loop.

Note: The “break” statement almost always needs an “if” condition to work properly.

Syntax:	
break	
Using break in while loop:	Using break in for loop:
<pre> while (condition): #statements if (condition): break #statements </pre> 	<pre> for var in sequence: #statements if (condition): break #statements </pre> 

Application:

<p>BREAK in WHILE loop :</p> <p>If the 'while' loop encounters an optional 'break', the loop simply exits even though the 'while' condition is True.</p>	<p>BREAK in FOR loop :</p> <p>If the 'for' loop encounters an optional 'break', the loop simply exits even though the 'for' sequence is not completed.</p>
<p>Application: Write a program to print 1-5 using a 'while' loop with 'break' to stop at 4.</p> <pre>i=1 # while loop with i = 1 to 3 while i <= 5: print(i) i += 1 if(i==4): break</pre> <p>Output:</p> <pre>1 2 3</pre> <p>Explanation: The while loop continued until it encountered 4 and then exited while loop. The i += 1 statement increments the value of i by 1 on each iteration of the loop.</p>	<p>Application: Write a program to print 1-5 using a 'for' loop with 'break' to stop at 4.</p> <pre># for loop with i = 1 to 3 for i in range (1,6): if(i==4): break print(i)</pre> <p>Output:</p> <pre>1 2 3</pre> <p>Explanation: The 'for' loop continued until it encountered 4 and then exited 'for' loop.</p>

2. "continue" statement**Definition:**

The "continue" statement forces the control to **skip the current iteration** and go to the next iteration of the loop.

- A. In "while" statement, the "continue" statement will directly **jump the execution control to "condition"**,
- B. In "for" statement, the "continue" statement will **jump the execution control to the next element in the given sequence**.

Syntax:	
<code>continue</code>	
Using continue in while loop:	Using continue in for loop:
<pre> while (condition): #statements if (condition): continue #statements </pre>	<pre> for var in sequence: #statements if (condition): continue #statements </pre>

Application:

<p>CONTINUE in WHILE loop:</p> <p>If the 'while' loop encounters an optional 'continue', the loop simply skips the current iteration and jumps to the 'condition' for next iteration.</p>	<p>CONTINUE in FOR loop :</p> <p>If the loop encounters an optional 'continue', the loop simply skips the current iteration and jumps to next iteration in the sequence.</p>
<p>Application: Program to print <u>even numbers between 1 and 5</u> using while loop & continue (skip) on odds</p> <pre> n = 1 while n < 5: n += 1 if (n % 2) != 0: continue print(n) </pre> <p>Output: 2 4</p> <p>Explanation: The while loop continued until 2. When it encountered 3, the value incremented to 4 and executed 'continue' to skip the iteration.</p>	<p>Application: Program to print <u>even numbers between 1 and 5</u> using 'for' loop & continue (skip) on odds</p> <pre> for n in range(1,6): if (n % 2) != 0: continue print(n) </pre> <p>Output: 2 4</p> <p>Explanation: The 'for' loop continued until 2. When it encountered 3, it executed 'continue' to skip 3 and continued with 4 in the sequence.</p>

3. "pass" statement

Nothing happens when the "pass" statement is executed. Hence, it is a **null** operation and is considered a placeholder for future code.

- Empty code is not allowed in loops, function definitions, class definitions, or if statements and causes errors in Python.
- So, "pass" statement is used to write empty loops, control statements, functions, or classes to avoid errors.

Syntax:

```
pass
```

Application:

<p>PASS in IF and WHILE loop</p> <pre>n=1 while (n<5): if (n==3): pass n += 1</pre>	<p>PASS in FOR loop</p> <pre>college = "Chalapathi" for i in college: pass</pre>
<p>PASS in Function</p> <pre>def func(): pass func()</pre>	<p>PASS in Class</p> <pre>class name: pass</pre>

Quick Reference

Comparison of Loops

'for' loop	'while' loop
Pre-Test or Entry controlled loop - Checks for LAST ELEMENT at TOP	Pre-Test or Entry controlled loop - CONDITION is specified at TOP
Sequence controlled loop (Known number of elements)	Event (or Condition) controlled loop
Use it when you know how many times to iterate	Use it when you don't know how many times to iterate
Repeats for all elements in a sequence, except the last one.	Repeats until a condition is met
Syntax: for var in sequence: loop statements else: (optional) block of statements	Syntax: Initialization (optional) while (Condition): Loop Block of statements Incr/Decr (optional) else: (optional) Block of statements
Example: for # 1-4, the 5 not included for i in range(1,5): print(i)	Example: while i = 1 # while loop for i = 1 to 5 while i <= 5: print(i) i += 1
Output: 1 2 3 4	Output: 1 2 3 4 5

Comparison of “break” and “continue” statements

break	continue
Used to terminate the loop	Used to SKIP current iteration and go to NEXT iteration
Control passed to outside the loop	Control passed to the beginning of the loop for next iteration
EXIT from control loop	Loop takes NEXT iteration
“break” is used in LOOPS (for, while)	“continue” is used in LOOPS (for, while)
Syntax: <pre>for var in sequence: #body of loop if(condition) break</pre>	Syntax: <pre>for var in sequence: #body of loop if(condition) continue</pre>
Example: <pre>for i in range(5): if(i==3): break print(i)</pre>	Example: <pre>for i in range(5): if(i==3): continue print(i)</pre>
Output: 0 1 2	Output: 0 1 2 4 5

Comparison of “break”, exit(), sys.exit(), quit()

break	exit () or sys.exit()	quit ()
<p>break is a keyword in Python; therefore it can't be used as a variable name.</p>	<p>exit() is a standard library function in Python. exit can be used as a variable name.</p> <p>sys module can also be used: import sys sys.exit()</p>	<p>quit() is a standard library function in Python. quit can be used as a variable name.</p>
<p>break causes an immediate termination from a loop (for, while) and jumps to the remaining program.</p>	<p>exit() terminates whole program execution.</p>	<p>quit() terminates whole program execution.</p>
<p>break transfers the control to outside the loop (for, while).</p>	<p>exit() returns the control to the operating system or another program that uses this one as a sub-process.</p>	
<p>Example of break // some code here before while loop while(true) ... if(condition) break; # some code after while loop</p>	<p>Example of exit() // some code here before while loop while(true) ... if(condition) exit() # some code after while loop</p>	<p>Example of quit() // some code here before while loop while(true) ... if(condition) quit() # some code after while loop</p>
<p>In the above code, break terminates the while loop and some code after the while loop will be executed after breaking the loop.</p>	<p>In the above code, when if(condition) returns True, exit() will be executed and the program will get terminated.</p>	<p>In the above code, when if(condition) returns True, quit() will be executed and the program will get terminated.</p>
<p>Conclusion: break is a statement that terminates loops and jumps to the next program statements.</p>	<p>Conclusion: exit() is a library function that causes the immediate termination of the entire program.</p>	<p>Conclusion: quit() is a library function that causes the immediate termination of the entire program.</p>

Application: Scenario based solution using while-else loop

```

''' Scenario: A team of players play a game. There is a qualified
score per the game. Each players can score and add up to total scored.
Aim: Find whether the team scored more or less of the qualified score.
Use while loop. '''
moreScores = 'yes'
totalScores = 0
player = 1
# Qualified Score per game
totalToQualify = int(input('What is the qualified score per game? '))
while moreScores == 'yes':
    # Get score per player
    scoresPerPlayer = int(input(f'Enter score for player {player}: '))
    totalScores += scoresPerPlayer
    player += 1
    # Ask if user wants to input another score
    moreScores = input('Do you want to enter score for another player? yes or
no : ')
else:
    print("*** End of the game ***")

#Calculate scores less/more than qualified score per game
if totalScores >= totalToQualify:
    print('Your team scored', abs(totalToQualify - totalScores), f'points
more than qualified score {totalToQualify} per game.')
elif totalScores <= totalToQualify:
    print(f'Your team scored', totalToQualify - totalScores, f'points less
than qualified score {totalToQualify} per game.')

```

Output:

```

What is the qualified score per game? 100
Enter score for player 1: 50
Do you want to enter score for another player? yes or no : yes
Enter score for player 2: 40
Do you want to enter score for another player? yes or no : yes
Enter score for player 3: 20
Do you want to enter score for another player? yes or no : no
*** End of the game ***

```

Your team scored 10 points more than qualified score 100 per game.