Control Structure of Algorithms- Repitition Structure

Repitition Structure

A repitition structure, also known as control structures is a block of one or more statements that are repeatedly executed as long as a specified condition remains true.

Usually, this loop has two important parts:

- An **expression** that is tested for true or false.
- A **statement** or **block of code** that is executed repeatedly if the expression evaluates to true.

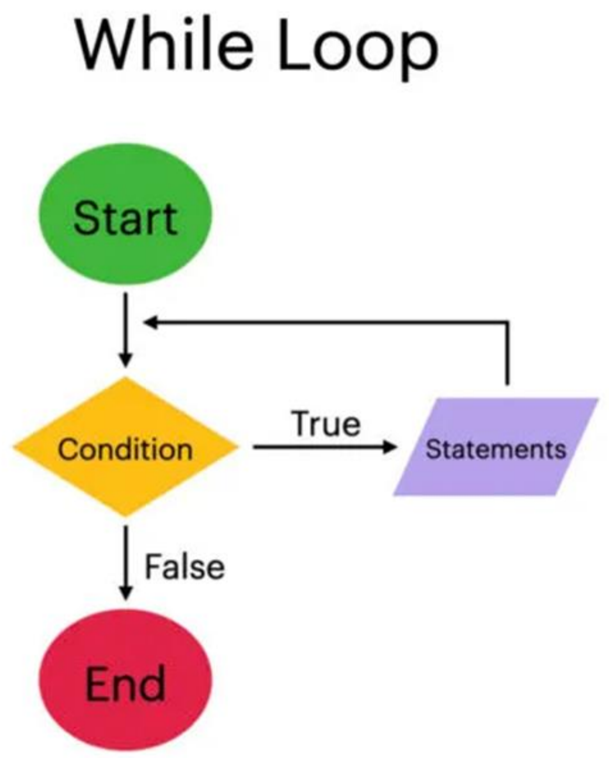
Two styles of repetition (or loops):

- Pre-test loop:
  - The condition is evaluated before the loop body is executed. Examples include while loops and for loops in most programming languages.
- Post-test loop:
  - The condition is evaluated after the loop body is executed. This ensures the loop body is executed at least once. Common in languages like C (do...while), but not available in Python.

Repitition Structure - Counters

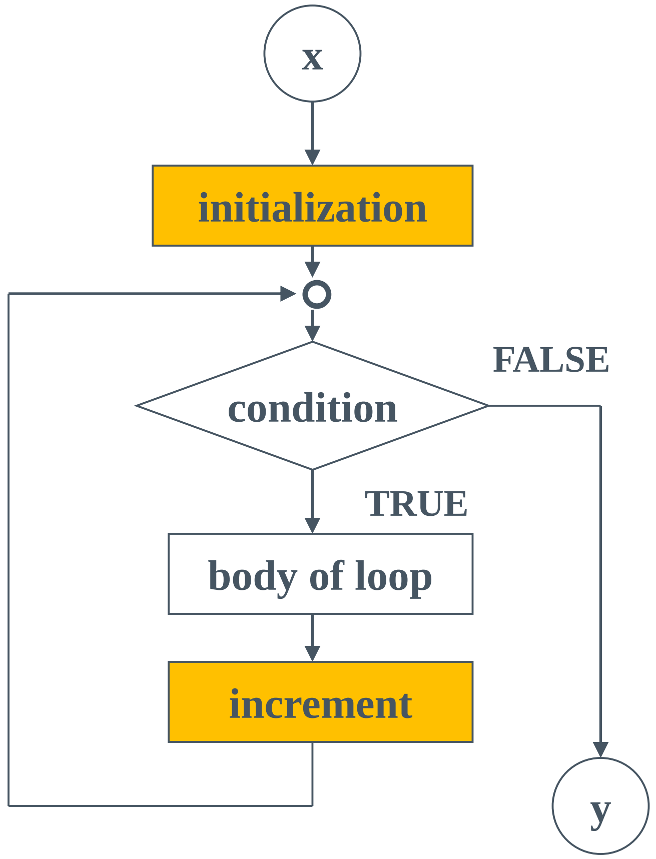
- Counters are used as **loop control variables** to regulate the execution of a loop.
- **Key Characteristics:**
  - **Increment or Decrement:** The counter is updated (increased or decreased) each time the loop repeats.
  - **Initialization:** The counter **must be initialized** before the loop begins to ensure proper execution and avoid infinite loops.

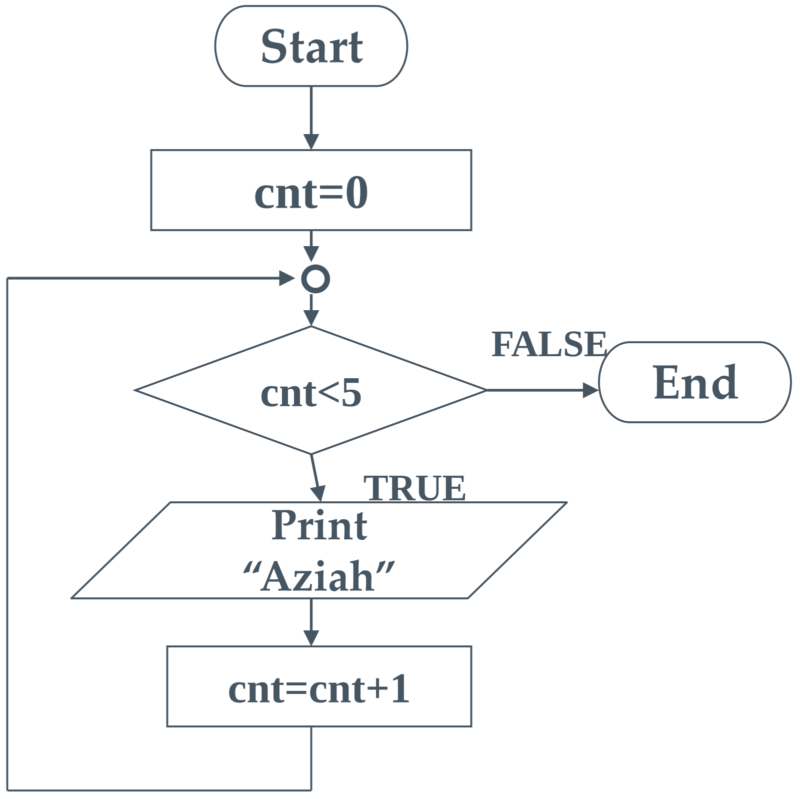
Repitition Structure- Pre test loop



```
While condition
      statement
end_while
```

Repetition Structure – Pre-test Loop (Example)





Steps in using a Counter-controlled loop :

- Initialization of counter
- Testing of counter value
- Updating of counter value during each iteration

Example:

```
counter = 1                               # Initialize the counter
n = 5                                     # Upper limit

while counter <= n:                       # Testing of counter value
    print(f"Counter: {counter}")
    counter += 1                          # Updating of the counter
```

Repitition Structures

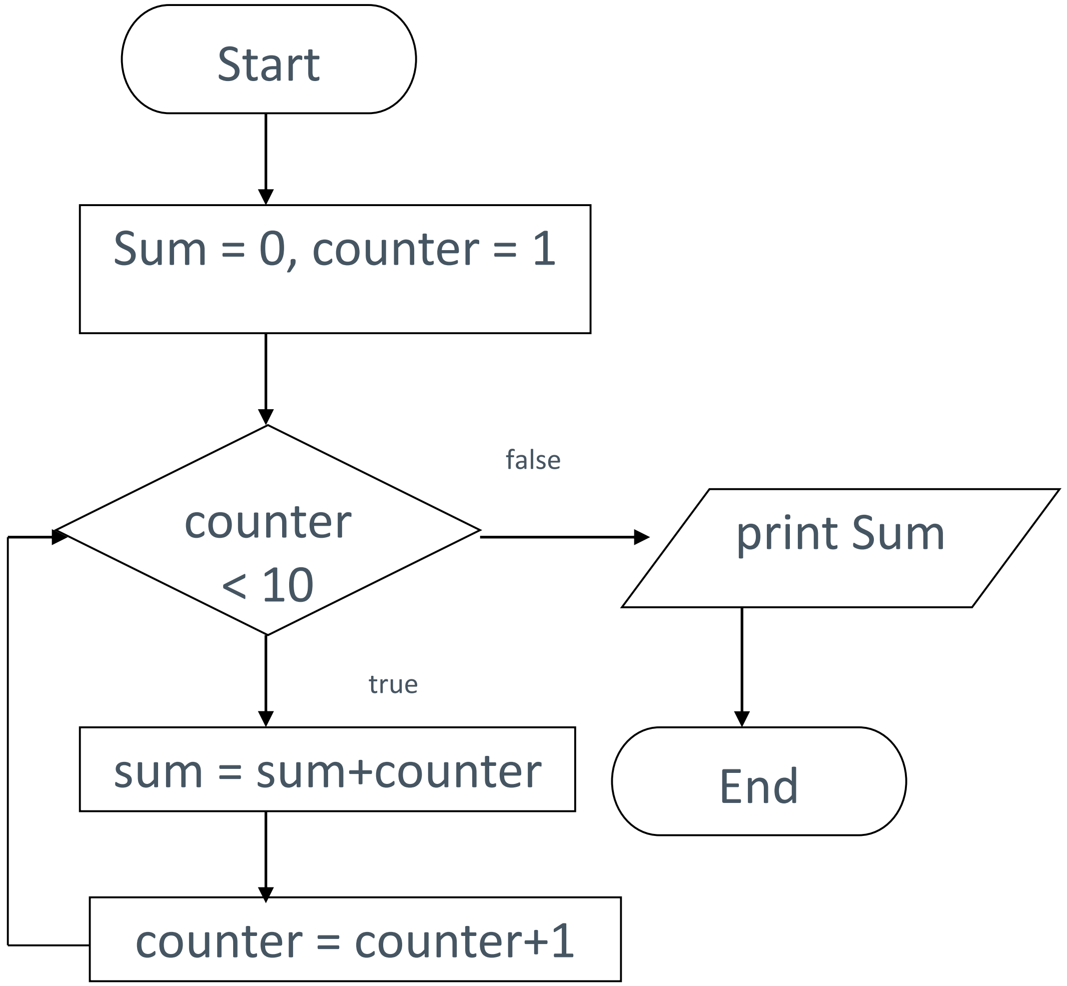
Write a program to compute a sum of the first 10 positive integers.

```
# Initialize the sum and counter
•  total_sum = 0
•  counter = 1

# While loop to iterate through the first 10 positive integers
while counter <= 10:
    total_sum += counter  # Add the current counter to the total sum
    counter += 1  # Increment the counter

# Display the result
print(f"The sum of the first 10 positive integers is: {total_sum}")
```

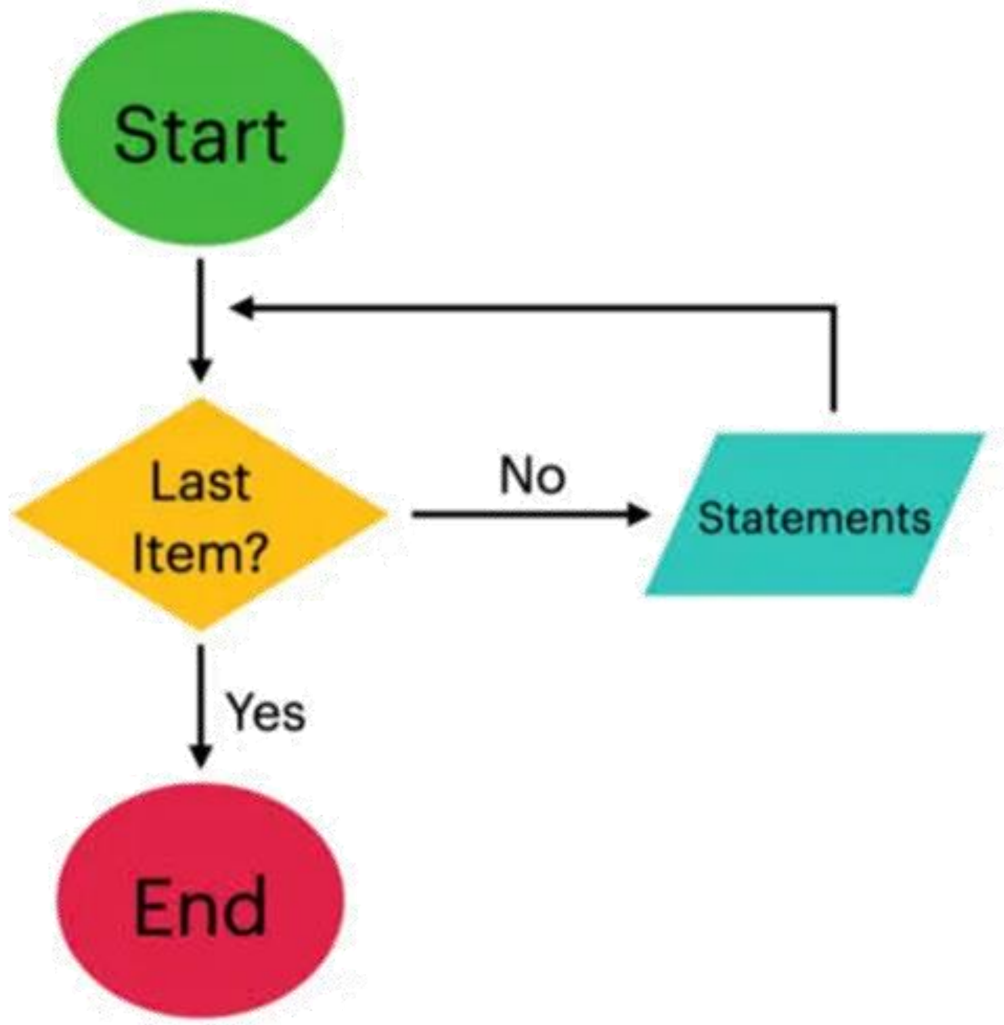
Pseudocode

```
Start
Set sum = 0
Set counter = 1
While (counter <= 10)
    sum = sum + counter
    counter = counter + 1
End_While
Display sum
End
```

Example - Flow Chart



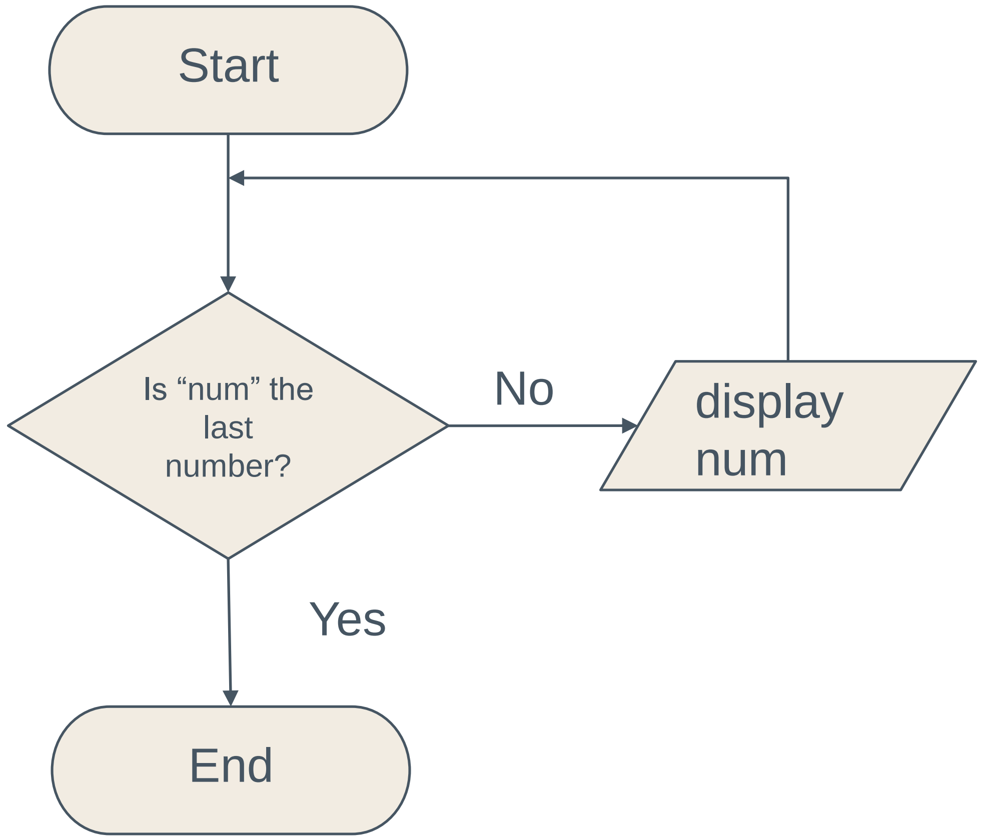
Repetition Control Structure – For Loop



```
For condition
      statement
```

```
for num in numbers
     display num
```

Example - Flowchart



Repetition Structure –
Letting the User Control a Loop

› Allow the user to control the repetition of a loop based on their input.

› Useful when the program needs to process a list of items, and the number of iterations is determined by the user.

› User is prompted before loop. Their input is used to control number of repetitions

Repetition Structure - Sentinels

- A **sentinel value** is a value in a list of values that indicates end of data.
- It acts as a marker to indicate the end of input data. This sentinel value is typically a value that can’t be confused with any valid input data.
- For example, using -999 as a sentinel value when entering test scores ensures that -999 is clearly understood to mean "end of input" rather than being a valid score.

Example

```
› scores = []
› sentinel_value = -999
› while True:
›     score = int(input(“vEnter a test score (or -999 to stop): "))
›     if score == sentinel_value:
›         break  # Exit the loop if sentinel value is entered
›     scores.append(score)
› print("Test scores entered:", scores)
```

Repetition Structure - Sentinels

```
Loop control by sentinel value

Start
sum = 0
read value
while(value != 0)
      sum = sum + value
      read value
print sum
End
```