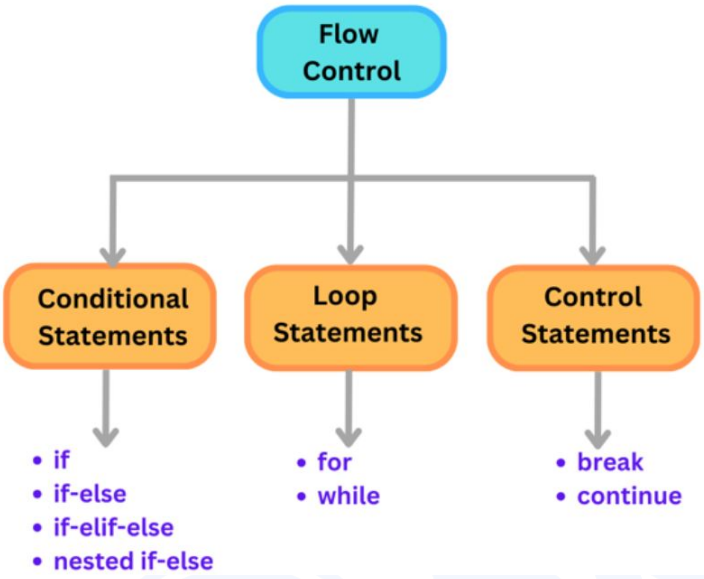
Lesson Plan:

Loops (While, For)

PW SKILLS

Topics to be covered:

1. Loop Statements
2. 'for' loop
3. Nested 'for' loop
4. 'While' loop



1. Loop Statements:

- Loop statements, often referred to as "loops," are a fundamental part of programming that allow you to repeatedly execute a block of code based on a specified condition.

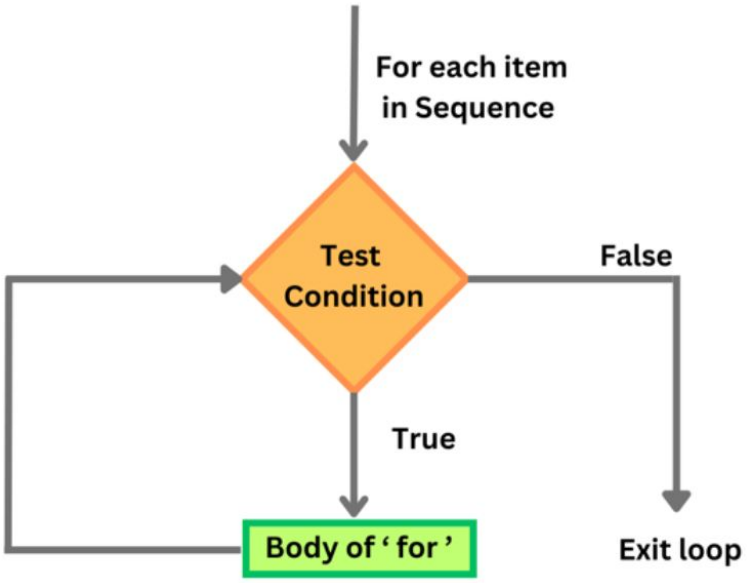
- Loops are used to automate repetitive tasks and iterate through data structures, making your code more efficient and less repetitive.

In Python, there are two main types of loop statements: for loops and while loops.

2. 'For' Loops:

- The `for` loop is used to iterate over a sequence of elements, such as a list or a string. It executes a block of code for each item in the sequence.

- The loop variable takes the value of each item in the sequence one by one. This loop is helpful when you know the number of iterations in advance or when you want to process each element of a collection.



Examples:

Iterating through a List:

```
fruits = ["apple", "banana", "cherry"]

for fruit in fruits:
    print("I like", fruit)
```

Generating a Number Sequence:

```
for i in range(1, 6):
    print("Number:", i)
```

3. Nested ‘for’ Loops:

- In Python, a nested for loop is one loop inside another loop. This is frequently used for activities where you have to repeatedly operate on values that fall within a certain range or iterate over elements in a grid or two-dimensional list.

A nested for loop is demonstrated here:

A nested for loop is demonstrated here:

Examples:

Pattern 1: Right Triangle

```
for i in range(5):
    for j in range(i + 1):
        print("*", end=" ")
    print()
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

Pattern 2: Square

```
for i in range(4):
    for j in range(4):
        print("*", end=" ")
    print()
```

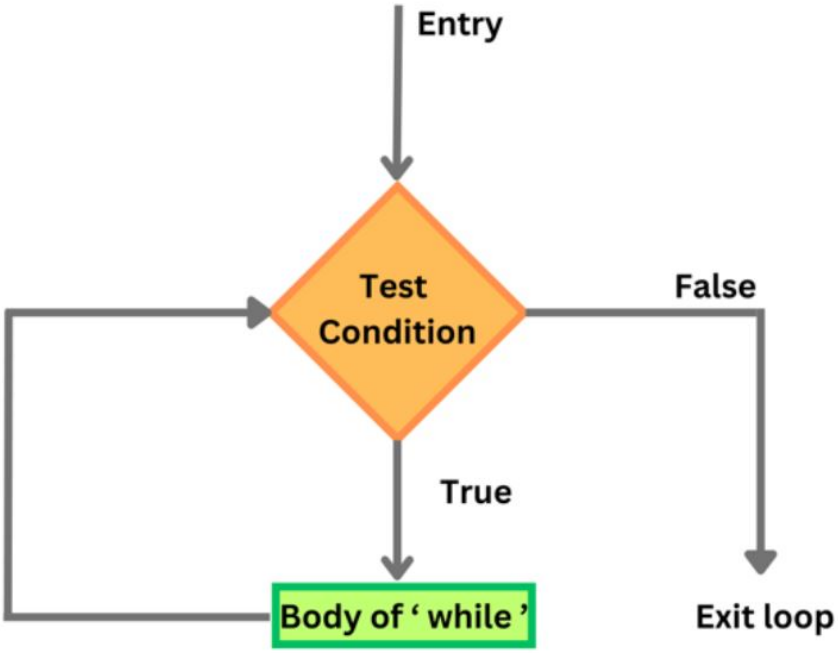
Output:

```
* * * *
* * * *
* * * *
* * * *
```

4. 'while' Loops:

- The `while` loop repeatedly executes a block of code as long as a given condition is true. The condition is checked before each iteration, and if it becomes false, the loop is exited.

- This allows for repetitive execution until a certain condition is met. Care must be taken to avoid infinite loops by ensuring the condition eventually becomes false.



Examples:

Counting Down:

```
count = 5

while count > 0:
    print(count)
    count -= 1
```

```
valid_input = False

while not valid_input:
    user_input = input("Enter 'yes' or 'no': ")
    if user_input.lower() in ["yes", "no"]:
        valid_input = True
    else:
        print("Invalid input. Please enter 'yes' or 'no'.")
```