TOPS Technology

Python Fundamentals

Presented By:

Nandni Vala

Looping (For, While)

1.Introduction to for and while loops.

- **for Loop**
- A for loop is used to iterate over a sequence (like a list, tuple, string, or range). It repeats the block of code for each item in the sequence.
- **Syntax:**

➢for variable in sequence:

➢ # Code to execute for each item

➢**Example:**

➢For i in range(1,n+1):

➢ Print("*",*i)

- **while Loops**
- A while loop repeats as long as a specified condition remains True. The condition is checked before executing the loop body.
- **Syntax:**
- while condition:
- # Code block
- **Example:**
- count = 5
- while count > 0:
- print("Countdown:", count)
- count -= 1

2.How loops work in Python.

- loops are used to execute a block of code repeatedly. They work by iterating through a sequence (like a list, string, or range) or by repeatedly checking a condition until it becomes False.
- **for Loop**
- The for loop in Python iterates over items in a sequence. For each item, the loop assigns it to a variable and executes the block of code inside the loop.
- **Steps:**
  - Take the first item from the sequence.
  - Assign it to the loop variable.
  - Execute the loop body.
  - Move to the next item in the sequence.
  - Repeat until all items are processed.
- **Example:**
- for number in [1, 2, 3]:
  - print("Number:", number)

- **while Loop**
- The while loop executes a block of code as long as a given condition evaluates to True.
- **Steps:**
  - Evaluate the condition.
  - If True, execute the loop body.
  - Recheck the condition.
  - Repeat until the condition becomes False.
- **Example:**

```
count = 0
while count < 3:
    print("Count:", count)
    count += 1
```

- Common Features in Loops:
- **break Statement**: Stops the loop immediately.

```
for i in range(5):
    if i == 3:
        break
    print("i:", i)
# Output: 0, 1, 2
```

- **continue Statement**: Skips the current iteration and moves to the next.

```
for i in range(5):
    if i == 3:
        continue
    print("i:", i)
# Output: 0, 1, 2, 4
```

3.Using loops with collections (lists, tuples, etc.).

- Using loops with collections like **lists**, **tuples**, and other iterable data structures is one of the most common operations in Python.
- **for Loop with Collections**
- **List**: Iterate over elements.
- **Example :**
- for item in [1, 2, 3]:
- print(item)
- **Tuple**: Similar to lists.
- **Example:**
- for item in (10, 20, 30):
- print(item)

- **Dictionary**: Loop through keys, values, or both.
- **Example :**
- my_dict = {"a": 1, "b": 2}
- for key, value in my_dict.items():
- print(key, value)
- **Set**: Iterate over unique, unordered elements.
- **Example :**
- for item in {1, 2, 3}:
- print(item)
- **while Loop with Collections**
- Use with an index or condition:
- nums = [10, 20, 30]
- i = 0
- while i < len(nums):
- print(nums[i])
- i += 1

- **Common Techniques**
- **enumerate()**: Get index and value
- **Example :**
- for i, value in enumerate(["a", "b", "c"]):
- print(i, value)