In this lecture, we explore the concept of loops and control statements in Python. Loops are used to execute a block of code repeatedly, while control statements like continue, break, and else help manage the flow of loops. Additionally, we discuss how to work with lists and dictionaries using loops, and the importance of indexing and enumerate functions.

Loops in Python

Loops are used to perform repetitive tasks efficiently. Python provides two types of loops:

- **For Loop:** Iterates over a sequence (e.g., list, tuple, dictionary).
- **While Loop:** Executes a block of code as long as a condition is true.

1- For Loop

For Loop Syntax

```
for item in sequence:
    # Code to execute
```

Example:

```
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)
```

Output:

```
1
2
3
4
5
```

Control Statements in Loops

Control statements are used to manage the flow of loops.
Continue Statement

- **Purpose:** Skips the current iteration and moves to the next one.
- **Use Case:** Ideal for bypassing specific conditions without stopping the loop.

Example:

```
for i in range(10):
    if i == 3:
        print("Skipping 3")
        continue
    print(i)
```

Output:

```
0
1
2
Skipping 3
4
5
6
7
8
9
```

Break Statement

- **Purpose:** Stops the loop entirely when a condition is met.

Example:

```
for i in range(10):
    if i == 8:
        print("Breaking at 8")
        break
    print(i)
```

Output:

```
0
1
2
3
4
5
6
7
Breaking at 8
```

Else Statement

- **Purpose:** Executes when the loop ends naturally (i.e., without a break statement).

Example:

```
for i in range(5):
    print(i)
else:
    print("Loop ended naturally")
```

Output:

```
0
1
2
3
4
Loop ended naturally
```

Working with Lists and Indexing

Lists are collections of items, and each item has an index (starting from 0).
Enumerate Function

- **Purpose:** Used to iterate over a list while accessing both the index and the item.

Example:

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(f"Index {index}: {fruit}")
```

Output:

```
Index 0: apple
Index 1: banana
Index 2: cherry
```

Manual Indexing Without enumerate, you can manually track the index.

Example:

```
fruits = ["apple", "banana", "cherry"]
index = 0
for fruit in fruits:
    print(f"Index {index}: {fruit}")
    index += 1
```

Output:

```
Index 0: apple
Index 1: banana
Index 2: cherry
```

Iterating Over Dictionaries

Dictionaries store data in key-value pairs. You can iterate over keys, values, or both.

Iterating Over Keys

```
person = {"name": "John", "age": 30, "city": "New York"}
for key in person:
    print(key)
```

Output:

```
name
age
city
```

Iterating Over Key-Value Pairs

```
for key, value in person.items():
    print(f"{key}: {value}")
```

Output:

```
name: John
age: 30
city: New York
```

Practical Example: Combining Loops and Control Statements

Here's a practical example that combines loops and control statements:

```
for i in range(10):
    if i == 3:
        print("Skipping 3")
        continue
    if i == 8:
        print("Breaking at 8")
        break
    print(i)
else:
```

```
    print("Loop ended naturally")
```

Output:

```
0
1
2
Skipping 3
4
5
6
7
Breaking at 8
```

2- While Loop

While Loop Syntax:

```
while condition:
    # Code to execute
```

Example:

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

Output:

```
1
2
3
4
5
```

While Loop with break:

```
i = 1
while i < 10:
    if i == 6:
        print("Breaking at 6")
        break
    print(i)
    i += 1
```

Output:

```
1
2
3
4
5
Breaking at 6
```

While Loop with continue:

```
i = 0
while i < 5:
    i += 1
    if i == 3:
        print("Skipping 3")
        continue
    print(i)
```

Output:

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

While Loop with else:

```
i = 1
while i <= 3:
    print(i)
    i += 1
else:
    print("Loop ended naturally")
```

Output:

```
1
2
3
Loop ended naturally
```

Best Practices for Using Loops

- Avoid Infinite Loops: Ensure the loop condition will eventually become false.
- Use enumerate for Indexing: Simplifies accessing both index and item.
- Leverage Control Statements: Use continue, break, and else to manage loop flow effectively.
- Practice with Different Scenarios: Experiment with loops in various conditions to build strong logic-building skills.

Final Thoughts

Loops and control statements are fundamental to Python programming, enabling developers to perform repetitive tasks efficiently and manage program flow effectively. By mastering for loops, while loops, and control statements like continue, break, and else, you can write cleaner and more efficient code. Additionally, understanding how to work with lists and dictionaries using loops is crucial for handling collections of data.

Key Takeaways:

- Use continue to skip specific iterations.
- Use break to exit a loop early.
- Use else to execute code after a loop ends naturally.
- Leverage enumerate for easy access to indices and items in lists.
- Practice loops with different scenarios to strengthen your programming skills.