Lecture Notes: Conditional Statements and Loops

Introduction

In this lecture, we will explore **conditional statements** and **loops**, two essential concepts in Python programming. These tools will help us write more efficient, dynamic, and interactive programs.

1. Conditional Statements

Conditional statements allow your code to make decisions based on certain conditions. Imagine a situation where you need to decide whether to carry an umbrella. You'll carry one if it's raining, but not otherwise. This decision-making process is similar to what happens in conditional statements.

1.1 If-Else Statement

The `if` statement checks if a condition is true. If it's true, the code inside the `if` block is executed. Otherwise, the `else` block is executed.

Syntax:

```
if condition:
    # Instructions when condition is true
else:
    # Instructions when condition is false
```

Example:

```
temperature = 25
if temperature > 30:
    print("It's a hot day.")
else:
    print("It's a nice day.")
```

Real-Life Example:

- If the store is open, you can buy groceries.
- Otherwise, you have to wait until it opens.

1.2 Elif Statement

What if you have more than two conditions? This is where the `elif` statement comes in. It allows for multiple conditions to be checked sequentially.

Syntax:

```
if condition1:
    # If condition1 is true
elif condition2:
    # If condition2 is true
else:
    # If neither condition is true
```

Example:

```
temperature = 25
if temperature > 30:
    print("It's a hot day.")
elif temperature > 20:
    print("It's a warm day.")
else:
    print("It's a cold day.")
```

Problem 1:
Write a Python program to check if a number is positive, negative, or zero using `if-else`.

Solution:

```
number = int(input("Enter a number: "))
if number > 0:
    print("The number is positive.")
elif number < 0:
    print("The number is negative.")
else:
    print("The number is zero.")
```

1.3 Nested If Statements

You can put an `if` statement inside another `if` statement to make more complex decisions.

Example:

```
age = 18
if age > 0:
    if age >= 18:
        print("You're an adult.")
    else:
        print("You're a minor.")
else:
    print("Invalid age.")
```

Trick Question:

If the number is `0`, what will the following code print?

```
if number > 0:
    print("Positive")
else:
    if number < 0:
        print("Negative")
    else:
        print("Zero")
```

2. Loops

Loops allow you to execute a block of code multiple times. Think of loops like brushing your teeth every day. You don't write a new instruction every day; instead, you repeat the same action.

2.1 For Loop

A `for` loop lets you iterate over a sequence (like a list, tuple, or string) and repeat the same block of code for each item.

Syntax:

```
for item in sequence:
    # Instructions
```

Example:

```
fruits = ['apple', 'banana', 'cherry']
for fruit in fruits:
    print(fruit)
```

Real-Life Example:

- Imagine you want to greet all your friends one by one. You could use a `for` loop to print a message for each friend.

2.2 For Loop with `range()`

The `range()` function generates a sequence of numbers. It's commonly used with `for` loops.

Syntax:

```
for i in range(start, stop, step):
    # Instructions
```

Example:

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

Problem 2:
What will the following code output?

```
for i in range(1, 10, 2):
    print(i)
```

Solution:
The code prints the numbers: 1, 3, 5, 7, 9.

2.3 While Loop

A `while` loop repeats as long as a condition is true.

Syntax:

```
while condition:
    # Instructions
```

Example:

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

Real-Life Example:

- Keep watering a plant while it is still thirsty.

Trick Question:
What will happen if you forget to update the `count` variable in the previous example?

3. Loop Control Statements

3.1 Break

The `break` statement exits a loop before it has finished.

Example:

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Output: 0, 1, 2 (The loop stops when `i` is 3.)

3.2 Continue

The `continue` statement skips the rest of the code in the current iteration and moves to the next one.

Example:

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

Output: 0, 1, 2, 4 (The loop skips `3`.)

Summary

- **Conditional Statements** help you make decisions in your code:
  - `if-else`: Basic two-way decisions.
  - `elif`: For multiple conditions.
  - Nested `if`: To handle more complex scenarios.
- **Loops** allow for repetition:
  - `for` loops for sequences like lists.
  - `while` loops for repeating actions as long as a condition is true.
- **Loop control** (`break`, `continue`) helps manage loop execution.

Trick Questions for Practice:

1. What will this code output?

```
for i in range(10, 1, -2):
    print(i)
```

2. Write a Python program to print all even numbers between 1 and 20 using a `for` loop.

Preview for Next Lecture:

In the next class, we'll dive deeper into **Nested Loops** and **Lists in Python**. Nested loops allow us to handle more complex repetition scenarios, and lists are a powerful data structure in Python that helps us organize and manipulate data efficiently.