Pre-Class Reading: Mastering Conditional Statements, While Loops, and For Loops in Python

Learning Objectives

In this session, you will:

- Gain a deep understanding of how conditional logic works in Python.
- Learn to write programs that make decisions based on user input or other conditions.
- Explore how the `if`, `elif`, and `else` statements allow for more flexible control over your program's flow.
- Practice using loops to execute code repeatedly, either for a set number of times (for loops) or until a condition is met (while loops).
- Discover how nested conditions and loops handle complex scenarios where multiple criteria or repetitions must be met.
- Understand the importance of breaking down complex logic using loops for more efficient and readable code.
- Apply your knowledge through real-world examples and exercises to reinforce your understanding.

Key Concepts and Vocabulary

1. If Statement:

A control structure that allows your program to execute a block of code if a certain condition is true. If the condition is false, the code inside the `if` block is skipped.
Example:

```
if condition:
    # code that runs if the condition is true
```

2. Else Statement:

Paired with an `if` statement, `else` allows you to specify what should happen if the condition is false. The `else` block will only execute if the `if` condition is false.
Example:

```
if condition:
    # code if true
else:
    # code if false
```

3. Elif Statement:

Short for "else if," this allows you to check multiple conditions. If the initial `if` condition is false, Python moves to check the `elif` conditions in sequence until one is true or defaults to the `else` block if none are true.
Example:

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

4. Nested Conditions:

This refers to placing an `if` statement inside another `if` or `else` block. It's useful for making decisions that depend on a sequence of conditions.
Example:

```
if condition1:
    if condition2:
        # code if both conditions are true
else:
    # code if condition1 is false
```

5. While Loop:

A loop that keeps running as long as a specified condition remains true. This is useful when you don’t know in advance how many times the code should be repeated.
Example:

```
while condition:
    # code that runs while the condition is true
```

6. For Loop:

A loop that iterates over a sequence (like a list or range of numbers) and executes a block of code a specific number of times or for each element in the sequence.
Example:

```
for item in sequence:
    # code that runs for each item in the sequence
```

7. Ternary Operator:

A shorthand way of writing an `if-else` statement. It’s used when you want to assign a value based on a condition and keep it concise.
Example:

```
value = "Yes" if condition else "No"
```

8. Indentation:

Unlike many other programming languages, Python relies on indentation to define blocks of code. This means it's crucial to maintain consistent indentation when writing conditional statements and loops.

Real-World Examples of Conditional Logic and Loops

Example 1: Traffic Lights

Traffic signals work using conditional logic. If a pedestrian presses the button to cross, the light changes after a brief delay. The conditional check is: "Has the button been pressed?" If yes, change the light; if no, continue with the regular cycle.

Code Analogy:

```
button_pressed = True

if button_pressed:
    print("Change the light to allow crossing.")
else:
    print("Continue with regular light cycle.")
```

Example 2: Repeating Tasks with a While Loop

Imagine you are building a password prompt that continues asking the user for a password until the correct one is entered.

Code Analogy:

```python
correct_password = "python123"
password = ""

while password != correct_password:
    password = input("Enter password: ")

print("Access granted!")
```

Example 3: Looping Over a List with a For Loop

Consider a situation where you want to send a welcome message to a list of new users.

Code Analogy:

```python
new_users = ["Alice", "Bob", "Charlie"]

for user in new_users:
    print(f"Welcome, {user}!")
```

Mini Code Snippets: Understanding Python’s Conditional Statements and Loops

1. Simple If-Else Statement:

```python
age = 20

if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible to vote.")
```

2. Using Elif for Multiple Conditions:

```python
temperature = 22

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.")
```

3. While Loop for Repeated Input:

```python
attempts = 0
while attempts < 3:
    password = input("Enter password: ")
    if password == "secret":
        print("Access granted")
        break
    attempts += 1
```

4. For Loop Iterating Over a Range:

```python
for i in range(5):
    print(f"This is loop iteration {i}")
```

Pre-Reading Questions or Simple Exercises

1. **Simple Condition**:
   Write a Python program that checks if a number entered by the user is positive, negative, or zero.
2. **Voting Age Check**:
   Modify the voting eligibility program to check if the person is eligible to vote based on both age and country. Use nested conditions for this.
3. **While Loop Practice**:
   Write a program that repeatedly asks the user to guess a number between 1 and 10. The loop should stop when they guess correctly.

4. **For Loop Practice**:
   Write a program that prints the squares of numbers from 1 to 10 using a `for` loop.

Common Mistakes or Misconceptions

1. **Forgetting Indentation**:
   Python uses indentation to define blocks of code. A common beginner mistake is to forget to indent code properly under `if` , `elif` , `else` , `while` , and `for` blocks.
2. **Incorrect Use of Elif and If**:
   Remember that `elif` is only checked if all preceding `if` or `elif` conditions are false. Sometimes beginners place `if` when they mean to use `elif` .
3. **Overusing Nested Loops and Conditions**:
   Nested loops and conditions are useful but can make code hard to read. Simplify where possible by breaking down logic or using boolean operators like `and` or `or` .

A Teaser for the Lecture

Here’s something to think about before class: What happens when you combine loops within loops, and how does Python manage these nested structures? We’ll dive into *nested loops* and explore how they can interact with *lists* to solve more complex problems.

In the next class, we’ll explore how nested loops can be used with data structures like lists to process complex data more efficiently.