Python section 7

Lesson 33

\*\*Section 7 Lesson 33: Understanding Python Loops and Repetition\*\*

\*\*Introduction\*\*

Good morning, everyone. My name is Ahmed Sami, and today we will dive into an essential topic in Python programming: loops. Specifically, we will analyze a Python script that demonstrates the use of the `while` loop and repetitive printing. By the end of this lecture, you will understand how loops work, how to control them, and how to apply them in your programs. Let’s begin by examining the code step by step.

---

\*\*Body\*\*

### Part 1: Repetition with `print` Statements

The code starts with a simple yet fundamental concept: repetition using the `print` statement.

```python

print("hello world")

print("hello world")

print("hello world")

print("hello world")

print("hello world")

```

Here, the phrase `"hello world"` is printed five times. While this approach works, it is not efficient. Imagine if we needed to print this phrase 100 or 1,000 times. Writing individual `print` statements would be tedious and impractical. This is where loops come into play.

---

### Part 2: Introduction to the `while` Loop

The `while` loop allows us to repeat a block of code as long as a specified condition is true. Let’s examine the first `while` loop in the code:

```python

i = 0

x = 100

while i < 10:

print("hello world")

i += 1

```

- \*\*Initialization\*\*: The variable `i` is initialized to `0`, and `x` is set to `100`. Note that `x` is not used in this loop, so it is irrelevant here.

- \*\*Condition\*\*: The loop continues as long as `i < 10`.

- \*\*Execution\*\*: Inside the loop, `"hello world"` is printed, and `i` is incremented by `1` using `i += 1`.

- \*\*Termination\*\*: The loop stops when `i` reaches `10`, as the condition `i < 10` becomes false.

This loop prints `"hello world"` exactly 10 times.

---

### Part 3: Modifying the Loop Condition

Next, the code introduces a modified version of the loop:

```python

i = 0

x = 100

while i < 3:

print("hello world")

i += 1

```

Here, the condition is changed to `i < 3`. As a result, the loop runs only 3 times, printing `"hello world"` three times. This demonstrates how changing the loop condition affects the number of iterations.

---

### Part 4: Printing the Loop Counter

The following loop introduces a new concept: printing the loop counter itself.

```python

i = 0

x = 100

while i < 3:

print(i)

i += 1

```

- Instead of printing `"hello world"`, this loop prints the value of `i`.

- The output will be:

```

0

1

2

```

- This is useful for tracking the progress of the loop or generating sequences of numbers.

---

### Part 5: Scaling the Loop

The next example scales the loop to run 100 times:

```python

i = 0

x = 100

while i < 100:

print(i)

i += 1

```

- The condition `i < 100` ensures the loop runs 100 times.

- The output will be numbers from `0` to `99`.

- This demonstrates how loops can handle large-scale repetition efficiently.

---

### Part 6: Using a Variable in the Loop Condition

Finally, the code shows how a variable can be used in the loop condition:

```python

i = 0

x = 100

while i < x:

print(i)

i += 1

```

- Here, the condition `i < x` uses the variable `x`, which is set to `100`.

- This loop behaves identically to the previous one, printing numbers from `0` to `99`.

- Using variables in conditions makes your code more flexible and dynamic.

---

\*\*Conclusion\*\*

In summary, loops are a powerful tool in Python that allow you to repeat code efficiently. The `while` loop, in particular, is useful when the number of iterations depends on a condition. By understanding how to initialize, control, and terminate loops, you can write more concise and effective programs.

I encourage you to experiment with the code we discussed today. Try changing the loop conditions, modifying the counter, or even combining loops with other Python features. If you have any questions, feel free to ask. Thank you for your attention, and I look forward to our next session.

---

\*\*End of Lecture\*\*

Python code for lesson 33

section 7 lesson 33 python code

print("hello world ")

print("hello world ")

print("hello world ")

print("hello world ")

print("hello world ")

new loop

while loop

i = 0

x = 100

while i < 10:

print("hello world")

i += 1

loop

i = 0

x = 100

while i < 3:

print("hello world")

i += 1

# loop

i = 0

x = 100

while i < 3:

print(i)

i += 1

# loop

i = 0

x = 100

while i < 100:

print(i)

i += 1

# loop

i = 0

x = 100

while i < x:

print(i)

i += 1

\*\*Section 7, Lesson 34: Understanding `break` and `continue` in Python Loops\*\*

\*\*Lecture by Dr. Ahmed Sami\*\*

---

### \*\*Introduction\*\*

Good morning, esteemed students. Today, in Section 7, Lesson 34, we will delve into two fundamental control flow statements in Python: `break` and `continue`. These statements are essential tools for managing the behavior of loops, allowing us to either terminate a loop prematurely or skip specific iterations. By the end of this lecture, you will understand how and when to use `break` and `continue` effectively in your programs. Let us begin by examining the provided code, which demonstrates these concepts in action.

---

### \*\*Body\*\*

#### \*\*1. The `break` Statement\*\*

The `break` statement is used to exit a loop prematurely, regardless of whether the loop condition has been fully satisfied. Let us analyze the first part of the code step by line:

```python

# using break

# break for breaking the loop

i = 0

x = 100

while i < x:

print(i)

i += 1

if i == 3:

break

```

1. \*\*Initialization\*\*:

- `i = 0`: We initialize a variable `i` with the value `0`. This will serve as our loop counter.

- `x = 100`: We define a variable `x` with the value `100`. This represents the upper limit for our loop.

2. \*\*Loop Condition\*\*:

- `while i < x`: The loop will continue executing as long as the value of `i` is less than `x` (which is `100`).

3. \*\*Loop Body\*\*:

- `print(i)`: Inside the loop, we print the current value of `i`.

- `i += 1`: We increment `i` by `1` in each iteration.

4. \*\*Condition for `break`\*\*:

- `if i == 3`: This condition checks if `i` has reached the value `3`.

- `break`: If the condition is met, the `break` statement is executed, causing the loop to terminate immediately.

\*\*Example Output\*\*:

```

0

1

2

```

\*\*Explanation\*\*:

The loop starts with `i = 0` and prints `0`. It then increments `i` to `1` and prints `1`. This process repeats until `i` reaches `3`. At that point, the `break` statement is triggered, and the loop terminates. Thus, the numbers `0`, `1`, and `2` are printed.

---

#### \*\*2. The `continue` Statement\*\*

The `continue` statement is used to skip the remaining code in the current iteration of a loop and proceed to the next iteration. Let us now examine the second part of the code:

```python

# using continue

# continue to continue the loop

i = 0

x = 100

while i < x:

print(i)

i += 1

if i == 3:

continue

```

1. \*\*Initialization\*\*:

- `i = 0`: The variable `i` is initialized to `0`.

- `x = 100`: The variable `x` is set to `100`.

2. \*\*Loop Condition\*\*:

- `while i < x`: The loop will run as long as `i` is less than `100`.

3. \*\*Loop Body\*\*:

- `print(i)`: The current value of `i` is printed.

- `i += 1`: The value of `i` is incremented by `1`.

4. \*\*Condition for `continue`\*\*:

- `if i == 3`: This condition checks if `i` equals `3`.

- `continue`: If the condition is met, the `continue` statement is executed, skipping any code below it for the current iteration and moving to the next iteration.

\*\*Example Output\*\*:

```

0

1

2

3

4

...

99

```

\*\*Explanation\*\*:

The loop starts with `i = 0` and prints `0`. It increments `i` to `1` and prints `1`. This process continues until `i` reaches `3`. At this point, the `continue` statement is executed, but since there is no code after it in this example, the loop simply proceeds to the next iteration. Thus, the output includes all numbers from `0` to `99`.

---

### \*\*Conclusion\*\*

In summary, the `break` and `continue` statements are powerful tools for controlling the flow of loops in Python. The `break` statement allows you to exit a loop prematurely, while the `continue` statement enables you to skip the remaining code in the current iteration and proceed to the next one. Understanding these concepts is crucial for writing efficient and flexible loops in your programs.

As you continue your journey in Python programming, I encourage you to experiment with these statements in different scenarios to deepen your understanding. Remember, mastery comes through practice. Thank you for your attention, and I wish you success in your coding endeavors.

---

\*\*End of Lecture\*\*

section 7 lesson 34 python code

# using break

# break for breaking the loop

i = 0

x = 100

while i < x:

print(i)

i += 1

if i == 3:

break

# using continue

# continue to continue the loop

i = 0

x = 100

while i < x:

print(i)

i += 1

if i == 3:

continue

### \*\*Section 7, Lesson 35: Understanding Python For Loops with Dr. Ahmed Sami\*\*

---

#### \*\*Introduction:\*\*

Welcome, students, to Section 7, Lesson 35! Today, we will dive into the world of Python \*\*for loops\*\*. For loops are one of the most fundamental concepts in programming, allowing us to iterate over sequences like lists, ranges, and more. By the end of this lecture, you will understand how to use for loops effectively, manipulate lists, and control the flow of your loops using `break` and `continue` statements. Let’s begin!

---

### \*\*Body:\*\*

#### \*\*1. Basic For Loop with `range()`\*\*

```python

# for loop

# from 0 to 9

for i in range(0, 10):

print(i)

```

- \*\*Explanation:\*\*

- `range(0, 10)` generates a sequence of numbers starting from 0 (inclusive) to 10 (exclusive). This means it will produce numbers from 0 to 9.

- The `for` loop iterates over each number in this sequence, and the variable `i` takes on each value one by one.

- `print(i)` outputs the current value of `i` to the console.

- \*\*Example:\*\*

- Output: `0 1 2 3 4 5 6 7 8 9`

---

#### \*\*2. For Loop with `range()` Including the End Value\*\*

```python

# for loop

# from 0 to 10

for i in range(0, 11):

print(i)

```

- \*\*Explanation:\*\*

- Here, `range(0, 11)` generates numbers from 0 to 10 (inclusive).

- The loop prints each number in this range.

- \*\*Example:\*\*

- Output: `0 1 2 3 4 5 6 7 8 9 10`

---

#### \*\*3. For Loop with List Indices\*\*

```python

# create a list that contains 4 items

names = ["ahmed", "jermaine", "sara", "ali"]

for i in range(len(names)):

print(i)

```

- \*\*Explanation:\*\*

- `names` is a list containing 4 items.

- `len(names)` returns the length of the list, which is 4.

- `range(len(names))` generates numbers from 0 to 3 (indices of the list).

- The loop prints the indices (`i`) of the list.

- \*\*Example:\*\*

- Output: `0 1 2 3`

---

#### \*\*4. For Loop Printing List Items Using Indices\*\*

```python

# create a list that contains 4 items

# printing the items in the list

names = ["ahmed", "jermaine", "sara", "ali"]

for i in range(len(names)):

print(names[i])

```

- \*\*Explanation:\*\*

- Similar to the previous example, but instead of printing the indices, it prints the actual items in the list using `names[i]`.

- \*\*Example:\*\*

- Output: `ahmed jermaine sara ali`

---

#### \*\*5. For Loop Iterating Directly Over List Items\*\*

```python

# the second way to create a for loop

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

```

- \*\*Explanation:\*\*

- This is a more Pythonic way to iterate over a list. Instead of using indices, the loop directly assigns each item in the list to the variable `person`.

- `print(person)` outputs each name in the list.

- \*\*Example:\*\*

- Output: `ahmed jermaine sara ali`

---

#### \*\*6. For Loop with `break` Statement\*\*

```python

# Use if statement with break

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "sara":

break

```

- \*\*Explanation:\*\*

- The loop iterates over the list `names`.

- When the condition `person == "sara"` is met, the `break` statement terminates the loop immediately.

- \*\*Example:\*\*

- Output: `ahmed jermaine sara`

---

#### \*\*7. For Loop with `break` on a Different Condition\*\*

```python

# change to jermaine

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

break

```

- \*\*Explanation:\*\*

- Similar to the previous example, but the loop breaks when `person == "jermaine"`.

- \*\*Example:\*\*

- Output: `ahmed jermaine`

---

#### \*\*8. For Loop with `continue` Statement\*\*

```python

# Use if statement with continue

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

continue

```

- \*\*Explanation:\*\*

- The `continue` statement skips the rest of the loop body for the current iteration if the condition is met.

- In this case, nothing happens because `person` never equals `"jermaine"` (note the typo in the code).

- \*\*Example:\*\*

- Output: `ahmed jermaine sara ali`

---

#### \*\*9. For Loop with `continue` and Skipping an Item\*\*

```python

# Opposite the loop

# Use if statement with continue

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

if person == "jermaine":

continue

print(person)

```

- \*\*Explanation:\*\*

- When `person == "jermaine"`, the `continue` statement skips the `print(person)` statement for that iteration.

- This effectively excludes `"jermaine"` from the output.

- \*\*Example:\*\*

- Output: `ahmed sara ali`

---

### \*\*Conclusion:\*\*

In this lecture, we explored the power of \*\*for loops\*\* in Python. We learned how to:

1. Iterate over a range of numbers using `range()`.

2. Loop through lists using indices and directly over items.

3. Control loop execution with `break` and `continue` statements.

These concepts are essential for writing efficient and readable Python code. Practice these examples, and you’ll master for loops in no time. Remember, programming is like exploring the universe—full of endless possibilities! Until next time, keep coding and exploring. Goodbye!

---

section 7 lesson 35 python code

# for loop

# from 0 to 9

for i in range(0, 10):

print(i)

# for loop

# from 0 to 10

for i in range(0, 11):

print(i)

# for loop

# create a list that contain 4 items

names = ["ahmed", "jermaine", "sara", "ali"]

for i in range(len(names)):

print(i)

# for loop

# create a list that contain 4 items

# printing the items in the list

names = ["ahmed","jermaine ", "sara", "ali"]

for i in range(len(names)):

print(names[i])

print("loop has ended ")

# for loop

# # create a list that contain 4 items

# the second way for create for loop

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

# for loop

# create a list that contain 4 items

# Use if statement with break

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "sara":

break

# for loop

# create a list that contain 4 items

# Use if statement with break

# change to jermaine

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

break

# for loop

# create a list that contain 4 items

# Use if statement with continue

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

continue

# for loop

# create a list that contain 4 items

# Opposite the loop

# Use if statement with continue

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

if person == "jermaine":

continue

print(person)

**Section 7 Lesson 35: Python Code Explanation**   
**Lecturer: Ahmed Sami**

In today’s lecture, we will explore various examples of Python for loops. Imagine our journey through time and space—each loop is like a new adventure, guiding us through sequences, lists, and decisions. We will break down the code line by line to clarify the concepts and illustrate how loop control works. This lecture is divided into three parts:

1. **Introduction**
2. **Body (Code Walk-Through)**
3. **Conclusion**

**1. Introduction**

1. We will review the structure and behavior of Python's for loops.
2. The examples include numeric ranges, iterating over list indices, directly iterating over list elements, and using control statements like break and continue.
3. Our goal is to understand not only how to iterate but also how to selectively exit or skip parts of a loop.

**2. Body (Code Walk-Through)**

*Example 1: Iterating Over a Numeric Range from 0 to 9*

1. The code starts with a comment explaining that a for loop will run from 0 to 9.

for i in range(0, 10):

print(i)

* **Line-by-Line Explanation:**
  1. The loop uses the range function with start value 0 and stop value 10. Remember, range goes from the starting number up to but not including the stop number, so it iterates over 0, 1, 2, \ldots, 9.
  2. For each iteration, the variable i is assigned the current value from the range.
  3. The print(i) statement displays the current value of i on the screen.

*Example 2: Iterating Over a Numeric Range from 0 to 10*

1. The next snippet is similar but extends the range:

for i in range(0, 11):

print(i)

* **Explanation:**
  1. Here, range(0, 11) creates a sequence from 0 up to 10 (since 11 is excluded).
  2. The loop prints each number, resulting in the output 0, 1, 2, \ldots, 10.

*Example 3: Iterating Over a List Using Indices (Printing Indices Only)*

1. A list named names is created containing four items.

names = ["ahmed", "jermaine", "sara", "ali"]

for i in range(len(names)):

print(i)

* **Explanation:**
  1. The list names holds four strings.
  2. len(names) returns 4; therefore, range(len(names)) produces the sequence 0, 1, 2, 3.
  3. The loop variable i takes each index, and print(i) outputs the index—this is useful when you need the position of the elements.

*Example 4: Iterating Over List Indices to Print the List Items*

1. The code is modified to print the actual items instead of just the indices.

names = ["ahmed", "jermaine", "sara", "ali"]

for i in range(len(names)):

print(names[i])

print("loop has ended ")

* **Explanation:**
  1. The list names is defined as before.
  2. For each index i in the range 0 to 3, print(names[i]) outputs the corresponding name.
  3. After the loop completes, the statement print("loop has ended ") indicates the loop’s termination.

*Example 5: Iterating Directly Over List Items*

1. A more Pythonic approach is presented:

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

* **Explanation:**
  1. Instead of iterating over indices, the loop directly extracts each element from the list names and assigns it to the variable person.
  2. The print(person) statement then outputs each name directly.
  3. This method is simpler and reduces the possibility of errors with index handling.

*Example 6: Using break to Exit a Loop Early (Condition: Name Equals "sara")*

1. The loop is combined with a condition that stops further iterations.

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "sara":

break

* **Explanation:**
  1. The loop iterates over the list names and prints each name.
  2. After printing, it checks whether person equals "sara".
  3. When the condition is met (i.e., once "sara" is printed), the break statement terminates the loop immediately. As a result, "ali" is never printed.

*Example 7: Using break with a Different Condition (Condition: Name Equals "jermaine")*

1. A subtle change in the condition demonstrates how loop termination behavior is affected.

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

break

* **Explanation:**
  1. The first name "ahmed" is printed.
  2. The next name "jermaine" is printed, and then the condition is met.
  3. The loop exits immediately after "jermaine" is printed, so "sara" and "ali" are skipped.

*Example 8: Using continue After a Print Statement (Condition: Name Equals "jermaine")*

1. The loop demonstrates the use of continue—a command that skips the rest of the code in the current iteration.

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

print(person)

if person == "jermaine":

continue

* **Explanation:**
  1. For each iteration, the name is printed first.
  2. When the person equals "jermaine", after printing, continue is executed. This skips any subsequent lines in that iteration.
  3. In this particular snippet, the continue has no visible effect because there are no statements after the condition. The same names are printed regardless.

*Example 9: Using continue to Skip Printing a Specific Name*

1. Here, the placement of the continue statement is such that it prevents printing the specified name.

names = ["ahmed", "jermaine", "sara", "ali"]

for person in names:

if person == "jermaine":

continue

print(person)

* **Explanation:**
  1. The loop checks if the current person equals "jermaine" before printing.
  2. If the condition is true, continue is invoked, which immediately jumps to the next iteration without executing print(person).
  3. Therefore, "jermaine" is skipped, and only "ahmed", "sara", and "ali" are printed.

**3. Conclusion**

1. **Summary of Key Concepts:**
   * **Numeric Ranges:** Using range(\, \) controls iterations from a (inclusive) to b (exclusive).
   * **List Iteration:** Iterating over list indices vs. directly accessing list elements demonstrates different approaches to loop structures.
   * **Loop Control:** The break statement halts a loop immediately when a condition is satisfied, while continue skips the rest of the current iteration.
2. **Application:**
   * These loop structures and control statements are fundamental in tasks such as data processing, simulation of processes, and algorithm implementation.
   * Understanding their behavior allows you to write precise and efficient code.
3. **Final Thought:**
   * Embrace the iterative journey, much like a voyage through the vast corridors of time and space—a lesson in control, order, and the elegance of Python programming.

This concludes our lecture on Python for loops in Section 7 Lesson 35. Reflect on how each control structure affects the flow of your program, and consider how you might apply these concepts in more complex coding scenarios.