رواد مصر الرقمية

وزارة الاتصـــــالات
وتكنولوجيا المعلومات

Python Programming

Session 2: (Loops with Practical Application)

Agenda:

1	For Loops and While Loops
2	Large Practical Application

For Loops and While Loops

For Loops and While Loops

Loops are fundamental control flow structures in Python that allow you to execute a block of code multiple times. This session will cover the for loop and while loop, which are used for iterating over sequences and executing code as long as a condition is true, respectively. Understanding these loops is crucial for handling repetitive tasks efficiently in programming.

For Loops

A for loop is used to iterate over a sequence (such as a list, tuple, string, or range) and execute a block of code for each item in the sequence.

For Loops

Examples

Iterating Over a List

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

```
apple
banana
cherry
```

This loop iterates over each item in the fruits list and prints it.

For Loops

Examples

Iterating Over a Range of Numbers

```
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

The range(5) function generates numbers from 0 to 4. The loop prints each number.

For Loops

Examples

Iterating Over a String

```
word = "Python"
for letter in word:
    print(letter)
```

```
P
y
t
h
o
n
```

This loop goes through each character in the string "Python" and prints it one by one.

For Loops

Examples

Using a For Loop with an Else Clause

```
for i in range(3):
    print(i)
else:
    print("Loop finished successfully.")
```

```
0
1
2
Loop finished successfully.
```

The loop prints numbers 0 to 2, and after the loop completes, the else clause executes

For Loops

Examples

Nested For Loop (Multiplication Table)

```
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} * {j} = {i * j}")
```

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
```

This nested loop generates a multiplication table for numbers 1 to 3.

While Loops

A while loop repeatedly executes a block of code as long as a specified condition remains true. It is often used when the number of iterations is not known beforehand.

While Loops

Examples

Basic While Loop

```
count = 0
while count < 5:
    print(count)
    count += 1
```

```
0
1
2
3
4
```

This loop prints the value of count and increments it by 1 each time.
It stops when count reaches 5.

While Loops

Examples

While Loop with Break Statement

```
count = 0
while count < 10:
    print(count)
    if count == 5:
        break
    count += 1
```

```
0
1
2
3
4
5
```

The loop prints numbers from 0 to 5. When count equals 5, the break statement stops the loop.

While Loops

Examples

While Loop with Continue Statement

```
count = 0
while count < 5:
    count += 1
    if count == 3:
        continue
    print(count)
```

```
1
2
4
5
```

This loop increments count by 1 in each iteration. When count is 3, continue skips printing and goes back to the loop.

While Loops

Examples

Infinite While Loop with a Break

```
while True:
    response = input("Type 'exit' to quit: ")
    if response == 'exit':
        break
```

```
Type 'exit' to quit: exit
```

This loop runs indefinitely until the user types "exit". When "exit" is typed, the break statement ends the loop.

While Loops

Examples

While Loop with an Else Clause

```
count = 0
while count < 3:
    print(count)
    count += 1
else:
    print("Loop finished.")
```

```
0
1
2
Loop finished.
```

The loop prints numbers from 0 to 2. When count reaches 3, the loop ends, and the else clause executes.

Summary

- **For Loops**: Ideal for iterating over a known sequence, such as lists, strings, or ranges. They provide a straightforward way to execute a block of code multiple times.
- **While Loops**: Useful when the number of iterations is not predetermined. They continue to execute as long as a condition remains true.
- **Break and Continue**: break exits the loop immediately, while continue skips to the next iteration.
- **Else with Loops**: Both for and while loops can have an else clause that executes when the loop completes naturally without hitting a break.

Large Practical Application

Finding the Maximum Value in a List

```
numbers = [3, 5, 2, 8, 1, 9, 4]
max_value = numbers[0]

for number in numbers:
    if number > max_value:
        max_value = number

print("The maximum value is:", max_value)
```

```
The maximum value is: 9
```

This for loop iterates over the numbers list to find the maximum value. It initializes max_value with the first element and updates it whenever it finds a larger number.

Counting Vowels in a String

```
text = "Hello, how are you?"
vowel_count = 0

for char in text:
    if char.lower() in 'aeiou':
        vowel_count += 1

print("Number of vowels:", vowel_count)
```

```
Number of vowels: 7
```

This for loop goes through each character in the string text and checks if it is a vowel. It increments the vowel_count for each vowel found.

Finding Prime Numbers within a Range

```python
start = 10
end = 20

for num in range(start, end + 1):
    if num > 1:
        for i in range(2, num):
            if num % i == 0:
                break
        else:
            print(num, "is a prime number")
```

```
11 is a prime number
13 is a prime number
17 is a prime number
19 is a prime number
```

The outer for loop iterates over numbers from start to end. The inner for loop checks if a number is divisible by any number other than 1 and itself. If it is, the break statement is executed, skipping to the next number.

Generating a Fibonacci Sequence

```python
n = 10
a, b = 0, 1
fibonacci_sequence = []

while len(fibonacci_sequence) < n:
    fibonacci_sequence.append(a)
    a, b = b, a + b

print("Fibonacci sequence:", fibonacci_sequence)
```

```
Fibonacci sequence: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

This while loop generates a Fibonacci sequence of length n. It appends each new term to the fibonacci_sequence list until it reaches the desired length.

Simulating a Basic ATM Withdrawal

```python
balance = 1000

while balance > 0:
    withdrawal = int(input("Enter amount to withdraw: "))
    if withdrawal <= balance:
        balance -= withdrawal
        print("Withdrawal successful. Remaining balance:", balance)
    else:
        print("Insufficient balance.")
    if balance == 0:
        print("Balance is zero. Exiting.")
        break
```

```
Enter amount to withdraw: 512
Withdrawal successful. Remaining balance: 488
Enter amount to withdraw: 400
Withdrawal successful. Remaining balance: 88
Enter amount to withdraw: 88
Withdrawal successful. Remaining balance: 0
Balance is zero. Exiting.
```

This while loop simulates an ATM withdrawal process. It checks if the withdrawal amount is less than or equal to the balance and updates the balance accordingly. If the balance reaches zero, it exits.

Finding Common Elements in Two Lists

```python
list1 = [1, 2, 3, 4, 5]
list2 = [4, 5, 6, 7, 8]
common_elements = []

for item in list1:
    if item in list2:
        common_elements.append(item)

print("Common elements:", common_elements)
```

```
Common elements: [4, 5]
```

This for loop checks each element in list1 to see if it exists in list2. If it does, the element is added to the common_elements list.

Calculating Factorial of a Number

```
num = 5
factorial = 1

for i in range(1, num + 1):
    factorial *= i

print("Factorial of", num, "is", factorial)
```

```
Factorial of 5 is 120
```

This for loop calculates the factorial of a given number (num). It multiplies each integer from 1 to num to compute the factorial.

Validating User Input with While Loop

```
while True:
    user_input = input("Enter a number greater than 10: ")
    if user_input.isdigit() and int(user_input) > 10:
        print("Thank you!")
        break
    else:
        print("Invalid input. Please try again.")
```

```
Enter a number greater than 10: 15
Thank you!
```

This while loop repeatedly asks the user for input until they enter a number greater than 10. The break statement exits the loop when valid input is provided.

Finding the Sum of Digits of a Number

```python
number = 1234
sum_of_digits = 0

while number > 0:
    digit = number % 10
    sum_of_digits += digit
    number //= 10

print("Sum of digits:", sum_of_digits)
```

```
Sum of digits: 10
```

This while loop calculates the sum of the digits of a given number. It extracts each digit using the modulus operator and adds it to sum_of_digits.

Drawing a Simple Pattern Using Loops

```
rows = 5

for i in range(1, rows + 1):
    for j in range(1, i + 1):
        print("*", end=" ")
    print("")
```

```
*
* *
* * *
* * * *
* * * * *
```

This nested for loop prints a simple right-angled triangle pattern using asterisks. The outer loop controls the number of rows, while the inner loop prints the stars.

Checking for Palindrome Strings

```
word = "madam"
is_palindrome = True

for i in range(len(word) // 2):
    if word[i] != word[-(i + 1)]:
        is_palindrome = False
        break

if is_palindrome:
    print(f"{word} is a palindrome.")
else:
    print(f"{word} is not a palindrome.")
```

```
madam is a palindrome.
```

This for loop checks if a string is a palindrome by comparing characters from the beginning and end moving towards the center. If any mismatch is found, it sets is_palindrome to False and breaks out of the loop.

Creating a Dictionary from Two Lists

```
keys = ["name", "age", "location"]
values = ["Alice", 30, "New York"]
dictionary = {}

for i in range(len(keys)):
    dictionary[keys[i]] = values[i]

print(dictionary)
```

```
{'name': 'Alice', 'age': 30, 'location': 'New York'}
```

This for loop iterates over the indices of the keys list and creates a dictionary by associating each key with the corresponding value from the values list.

Simulating a Simple Password Check

```python
correct_password = "python123"
attempts = 3

while attempts > 0:
    password = input("Enter your password: ")
    if password == correct_password:
        print("Access granted.")
        break
    else:
        attempts -= 1
        print(f"Incorrect password. You have {attempts} attempts left.")
else:
    print("Access denied. You have used all attempts.")
```

```
Enter your password: fs5fs44
Incorrect password. You have 2 attempts left.
Enter your password: fsfsf15
Incorrect password. You have 1 attempts left.
Enter your password: python123
Access granted.
```

This while loop simulates a password check, allowing the user up to three attempts to enter the correct password. If the correct password is entered, the loop breaks. Otherwise, it decrements the attempts and continues.

Generating a Multiplication Table for a Given Number

```python
number = 7
limit = 10

for i in range(1, limit + 1):
    print(f"{number} x {i} = {number * i}")
```

```
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

This for loop generates a multiplication table for the number (7 in this case) up to the specified limit (10). It multiplies the number by each value from 1 to limit and prints the result.