Learn Python Programming from Scratch

Topic: For Loops in Python

1. What are For Loops?

For loops are used to iterate over sequences (like lists, strings, tuples) or other iterable objects. Think of them as a way to repeat code for each item in a collection, making your programs more efficient and less repetitive. For loops are perfect when you know in advance how many times you want to repeat something.

2. Basic For Loop Syntax

The basic syntax of a for loop involves iterating over items in a sequence.

In [1]:
```python
# Basic for loop syntax
fruits = ["apple", "banana", "orange"]

for fruit in fruits:
    print(f"I like {fruit}")

print("Loop finished!")
```

```
I like apple
I like banana
I like orange
Loop finished!
```

3. Using the range() Function

The `range()` function generates a sequence of numbers, commonly used with for loops when you need to repeat code a specific number of times.

In [2]:
```python
# Using range() function
# range(5) generates numbers 0, 1, 2, 3, 4
for i in range(5):
    print(f"Number: {i}")

# range(start, stop, step)
for i in range(2, 10, 2):
    print(f"Even number: {i}")
```

```
Number: 0
Number: 1
Number: 2
Number: 3
Number: 4
Even number: 2
Even number: 4
Even number: 6
Even number: 8
```

4. Iterating Over Different Data Types

For loops can iterate over various data types including strings, lists, tuples, dictionaries, and sets.

In [3]:
```
# Iterating over a string
word = "Python"
for letter in word:
    print(letter)

# Iterating over a list
numbers = [1, 2, 3, 4, 5]
total = 0
for num in numbers:
    total += num
print(f"Sum: {total}")
```

```
P
y
t
h
o
n
Sum: 15
```

5. The enumerate() Function

`enumerate()` returns both the index and value of items in a sequence, which is useful when you need both position and value.

In [4]:
```
# Using enumerate() to get index and value
colors = ["red", "green", "blue"]

for index, color in enumerate(colors):
    print(f"Index {index}: {color}")

# Starting enumerate from 1 instead of 0
for position, color in enumerate(colors, start=1):
    print(f"Position {position}: {color}")
```

```
Index 0: red
Index 1: green
Index 2: blue
Position 1: red
Position 2: green
Position 3: blue
```

6. Nested For Loops

You can place for loops inside other for loops to handle multi-dimensional data or create patterns.

In [5]:
```
# Nested for loops example
for i in range(1, 4):
```

```
    for j in range(1, 4):
        product = i * j
        print(f"{i} × {j} = {product}")
    print()  # Empty line after each row
```

```
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
```

Exercises

1. Write a program to print numbers from 1 to 10 using a for loop.
2. Create a program that calculates the sum of all numbers in a list.
3. Print all even numbers between 1 and 20.
4. Create a multiplication table for any number using nested loops.
5. Count the number of vowels in a given string.

Practical Examples

Let's explore some practical examples of working with for loops in Python. These examples demonstrate real-world applications of iteration and repetition.

Student Grade Analysis System

Here's a practical example of using for loops to analyze student grades and calculate various statistics.

In [6]:
```
# Student grade analysis using for loops

# Student data with names and their scores
students = ["Alice", "Bob", "Charlie", "Diana", "Eve"]
scores = [85, 92, 78, 96, 88]

print("Student Grade Analysis")
print("=" * 30)

# Calculate and display individual grades
total_score = 0
highest_score = 0
lowest_score = 100
top_student = ""
lowest_student = ""
```

```
for i in range(len(students)):
    student = students[i]
    score = scores[i]

    # Determine letter grade
    if score >= 90:
        letter_grade = "A"
    elif score >= 80:
        letter_grade = "B"
    elif score >= 70:
        letter_grade = "C"
    elif score >= 60:
        letter_grade = "D"
    else:
        letter_grade = "F"

    print(f"{student}: {score} points - Grade {letter_grade}")

    # Update statistics
    total_score += score

    if score > highest_score:
        highest_score = score
        top_student = student

    if score < lowest_score:
        lowest_score = score
        lowest_student = student

# Calculate class statistics
class_average = total_score / len(students)

print("\nClass Statistics:")
print(f"Class Average: {class_average:.1f}")
print(f"Highest Score: {highest_score} ({top_student})")
print(f"Lowest Score: {lowest_score} ({lowest_student})")
```

```
Student Grade Analysis
==============================
Alice: 85 points - Grade B
Bob: 92 points - Grade A
Charlie: 78 points - Grade C
Diana: 96 points - Grade A
Eve: 88 points - Grade B

Class Statistics:
Class Average: 87.8
Highest Score: 96 (Diana)
Lowest Score: 78 (Charlie)
```

Pattern Generation and Mathematical Calculations

This example demonstrates using nested for loops to create patterns and perform mathematical calculations.

In [7]:
```
# Pattern generation and mathematical calculations
```

```python
print("Star Pattern Generation")
print("=" * 25)

# Generate a right triangle pattern
rows = 5
for i in range(1, rows + 1):
    for j in range(i):
        print("*", end="")
    print()  # New line after each row

print("\nMultiplication Table")
print("=" * 20)

# Generate multiplication table (5x5)
print("   ", end="")
for j in range(1, 6):
    print(f"{j:4}", end="")
print()

for i in range(1, 6):
    print(f"{i}: ", end="")
    for j in range(1, 6):
        product = i * j
        print(f"{product:4}", end="")
    print()

print("\nNumber Analysis")
print("=" * 17)

# Analyze numbers in a range
numbers = [12, 15, 18, 21, 24, 27, 30]
even_count = 0
odd_count = 0
divisible_by_3 = []

for number in numbers:
    if number % 2 == 0:
        even_count += 1
    else:
        odd_count += 1

    if number % 3 == 0:
        divisible_by_3.append(number)

print(f"Numbers analyzed: {numbers}")
print(f"Even numbers: {even_count}")
print(f"Odd numbers: {odd_count}")
print(f"Divisible by 3: {divisible_by_3}")
```

```
Star Pattern Generation
========================
*
**
***
****
*****

Multiplication Table
====================
      1   2   3   4   5
1:    1   2   3   4   5
2:    2   4   6   8  10
3:    3   6   9  12  15
4:    4   8  12  16  20
5:    5  10  15  20  25

Number Analysis
================
Numbers analyzed: [12, 15, 18, 21, 24, 27, 30]
Even numbers: 4
Odd numbers: 3
Divisible by 3: [12, 15, 18, 21, 24, 27, 30]
```

Key For Loop Rules to Remember

Let's review the important rules and best practices for working with for loops:

- Use for loops when you need to iterate over a known sequence or range
- The loop variable takes each value from the sequence in order
- Use `range()` to generate number sequences for counting loops
- Use `enumerate()` when you need both index and value during iteration
- Nested loops multiply the number of iterations (outer × inner)
- Use meaningful variable names for loop variables (not just i, j, k)
- Be careful with nested loops as they can impact performance with large data
- Use `break` to exit a loop early and `continue` to skip to the next iteration
- Indent the loop body properly (4 spaces is the Python standard)
- Consider using list comprehensions for simple transformations
- Test your loops with different data sizes to ensure they work correctly

In [8]:
```python
# Examples of good for loop practices

# Example 1: Processing a shopping list with prices
shopping_list = [
    ("Milk", 3.50),
    ("Bread", 2.25),
    ("Eggs", 4.00),
    ("Cheese", 5.75),
    ("Apples", 3.25)
]

print("Shopping List Analysis")
print("=" * 25)
```

```
total_cost = 0
expensive_items = []
cheap_items = []

for item_name, price in shopping_list:
    print(f"{item_name}: ${price:.2f}")
    total_cost += price

    if price > 4.00:
        expensive_items.append(item_name)
    elif price < 3.00:
        cheap_items.append(item_name)

print(f"\nTotal Cost: ${total_cost:.2f}")
print(f"Expensive Items (>$4.00): {expensive_items}")
print(f"Budget Items (<$3.00): {cheap_items}")
```

```
Shopping List Analysis
==========================
Milk: $3.50
Bread: $2.25
Eggs: $4.00
Cheese: $5.75
Apples: $3.25

Total Cost: $18.75
Expensive Items (>$4.00): ['Cheese']
Budget Items (<$3.00): ['Bread']
```

In [9]:
```
# Example 2: Text analysis
text = "Python programming is fun and powerful"
words = text.split()

print(f"\nText Analysis")
print("=" * 15)

word_lengths = []
long_words = []

for word in words:
    length = len(word)
    word_lengths.append(length)

    if length > 6:
        long_words.append(word)

average_length = sum(word_lengths) / len(word_lengths)
print(f"Total words: {len(words)}")
print(f"Average word length: {average_length:.1f}")
print(f"Long words (>6 chars): {long_words}")
```

```
Text Analysis
===============
Total words: 6
Average word length: 5.5
Long words (>6 chars): ['programming', 'powerful']
```

Course Information

Learn Python Programming from Scratch

Author: Prakash Ukhalkar

Topic: Python Control Flow - For Loops

Built with ❤️ for the Python learning community