Introduction to Python Programming

Chapter 6: Loops

Introduction

Loops allow you to repeat code multiple times without writing it over and over. This is one of the most powerful features in programming - it's where automation really begins! In this chapter, we'll learn about two types of loops: **for loops** and **while loops**.

---

1. For Loops vs While Loops: The Big Picture

Before we dive into details, let's understand when to use each type of loop:

For Loops

Use when you know (or can determine) how many times to repeat

- Count from 1 to 10
- Process each item in a list
- Repeat something a specific number of times
- Even if a user provides the number, if you know it before the loop starts, use a for loop

While Loops

Use when repetition depends on a condition

- Keep asking until user enters valid input
- Continue until some condition is met
- Run a game until player chooses to quit
- When you don't know how many iterations you'll need

Key difference: For loops have a predetermined number of iterations. While loops continue based on a condition that might change during execution.

---

2. The For Loop and Range

What is Range?

Before we can understand for loops fully, we need to understand `range()`.

Range creates a sequence of numbers that we can iterate through. Think of it as generating a list of numbers that the loop will go through, one at a time.

Range with Stop Value Only

The simplest form: `range(stop)`

```
range(5)
# Generates: 0, 1, 2, 3, 4
```

Important observations:

1. It starts at 0 by default (computers like starting at 0!)
2. It goes UP TO but DOES NOT INCLUDE the stop value
3. So `range(5)` gives you 5 numbers: 0, 1, 2, 3, 4

Why start at 0? It seems strange at first, but starting at 0 is incredibly useful for accessing list items, counting iterations, and many programming tasks. You'll see why this is brilliant as we progress!

Range with Start and Stop Values

Format: `range(start, stop)`

```
range(3, 8)
# Generates: 3, 4, 5, 6, 7
```

Key points:

- First number is the START (included)
- Second number is the STOP (NOT included)
- `range(3, 8)` gives: 3, 4, 5, 6, 7

Range with Start, Stop, and Step

Format: `range(start, stop, step)`

The step determines how much to increase by each time.

```
range(0, 10, 2)
# Generates: 0, 2, 4, 6, 8

range(1, 11, 2)
# Generates: 1, 3, 5, 7, 9

range(10, 0, -1)
# Generates: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

Default values:

- Default start: 0
- Default step: 1
- Stop is ALWAYS required

---

3. Using For Loops with Range

Basic For Loop Structure

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

Output:

```
0
1
2
3
4
```

How It Works

Think of the range as creating a list: [0, 1, 2, 3, 4]

1. First time through: `i` is 0, run the code block
2. Second time through: `i` is 1, run the code block
3. Third time through: `i` is 2, run the code block
4. And so on...
5. When we run out of numbers in the range, exit the loop

The variable `i` (or whatever you name it) temporarily holds the current value from the range.

```
for count in range(3):
    print(f"Loop iteration: {count}")
```

Output:

```
Loop iteration: 0
Loop iteration: 1
Loop iteration: 2
```

For Loop Examples

Example 1: Counting 0 to 9

```
for i in range(10):
    print(i)
# Prints: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

Example 2: Counting 5 to 10

```
for num in range(5, 11):
    print(num)
# Prints: 5, 6, 7, 8, 9, 10
```

Notice: To get 10 included, we use 11 as the stop value!

Example 3: Counting by 2s

```
for i in range(0, 20, 2):
    print(i)
# Prints: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18
```

Example 4: Counting backwards

```
for i in range(10, 0, -1):
    print(i)
# Prints: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
```

Example 5: Using user input

```
times = int(input("How many times to print hello? "))

for i in range(times):
    print("Hello!")
```

Even though the user determines the number, we KNOW how many iterations before the loop starts - perfect for a for loop!

Important Note About Range

Range is not the only way to use for loops! Later in this course, we'll learn about iterating through lists, strings, and other collections. For now, we're focusing on range-based for loops to build a strong foundation.

---

4. While Loops

While loops continue as long as a condition is True. They're perfect when you don't know exactly how many times you'll loop.

Basic While Loop Structure

```
while condition:
    # code block
    # this runs as long as condition is True
```

How it works:

1. Check the condition
2. If True: run the code block, then go back to step 1
3. If False: exit the loop and continue with the rest of the program

While Loop Examples

Example 1: Count to 5

```
count = 1

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

print("Done!")
```

Output:

```
1
2
3
4
5
Done!
```

How it works:

- Start: count is 1, condition (1 <= 5) is True → print 1, count becomes 2
- Next: count is 2, condition (2 <= 5) is True → print 2, count becomes 3
- ... continues ...
- Finally: count is 6, condition (6 <= 5) is False → exit loop

Example 2: Keep asking until valid input

```
password = ""

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

print("Access granted!")
```

This keeps looping until the user enters "secret". We don't know how many attempts it will take!

Example 3: Sum numbers until user enters 0

```
total = 0
number = int(input("Enter a number (0 to stop): "))

while number != 0:
    total = total + number
    number = int(input("Enter a number (0 to stop): "))

print(f"Total: {total}")
```

Comparing For and While

Same task, different approaches:

```python
# Using for loop - we know iterations
for i in range(5):
    print("Hello")

# Using while loop - also works
count = 0
while count < 5:
    print("Hello")
    count = count + 1
```

Both print "Hello" 5 times, but:

- For loop: Number of iterations is clear from the range
- While loop: We manage the counter ourselves

General rule: If you know the number of iterations, use a for loop. It's cleaner!

---

5. Break Statement

The `break` statement immediately exits a loop, regardless of the condition.

Using Break in For Loops

```python
for i in range(10):
    if i == 5:
        break
    print(i)

print("Loop ended")
```

Output:

```
0
1
2
3
4
Loop ended
```

When `i` equals 5, `break` exits the loop immediately. The print(i) doesn't run for 5, and we never get to 6, 7, 8, 9.

Using Break in While Loops

Break is especially useful in while loops:

```
while True:
    user_input = input("Enter 'quit' to exit: ")

    if user_input == "quit":
        break

    print(f"You entered: {user_input}")

print("Goodbye!")
```

The While True Pattern

`while True` creates an infinite loop - it will run forever unless we use `break` to exit.

Why use this? It's perfect for menus and game loops where you want to keep running until a specific exit condition:

```
while True:
    print("\n=== Menu ===")
    print("1. Play game")
    print("2. View scores")
    print("3. Quit")

    choice = input("Enter choice: ")

    if choice == "1":
        print("Starting game...")
    elif choice == "2":
        print("Viewing scores...")
    elif choice == "3":
        print("Thanks for playing!")
        break
    else:
        print("Invalid choice")
```

This menu keeps running until the user chooses option 3. The break is the ONLY way out.

Important: If you use `while True`, you MUST have a `break` somewhere, or your program will run forever!

---

6. Continue Statement

The `continue` statement skips the rest of the current iteration and goes to the next one.

Using Continue in For Loops

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

Output:

```
1
3
5
7
9
```

What happens:

- When `i` is even (0, 2, 4, 6, 8), the continue statement skips the print
- The loop goes to the next iteration
- Only odd numbers get printed

Using Continue in While Loops

```
count = 0

while count < 5:
    count = count + 1

    if count == 3:
        continue

    print(count)
```

Output:

```
1
2
4
5
```

When count is 3, continue skips the print and goes back to the while condition.

Break vs Continue

Break: Exit the loop completely
Continue: Skip to the next iteration

```python
# Demonstrating the difference
print("Break example:")
for i in range(5):
    if i == 3:
        break
    print(i)
# Output: 0, 1, 2

print("\nContinue example:")
for i in range(5):
    if i == 3:
        continue
    print(i)
# Output: 0, 1, 2, 4
```

---

7. Break and Continue in Nested Loops

When you have loops inside loops (nested loops), break and continue only affect the INNERMOST loop they're in.

Break in Nested Loops

```python
for i in range(3):
    print(f"Outer loop: {i}")

    for j in range(3):
        if j == 2:
            break
        print(f"  Inner loop: {j}")

    print("  Inner loop ended")
```

Output:

```
Outer loop: 0
  Inner loop: 0
  Inner loop: 1
  Inner loop ended
Outer loop: 1
  Inner loop: 0
  Inner loop: 1
  Inner loop ended
Outer loop: 2
  Inner loop: 0
  Inner loop: 1
  Inner loop ended
```

Notice: The break only exits the inner loop. The outer loop continues normally.

Continue in Nested Loops

```python
for i in range(3):
    for j in range(3):
        if j == 1:
            continue
        print(f"i={i}, j={j}")
```

Output:

```
i=0, j=0
i=0, j=2
i=1, j=0
i=1, j=2
i=2, j=0
i=2, j=2
```

The continue skips j=1 in the inner loop, but the outer loop is unaffected.

Important Concept

Break and continue affect only the loop they're directly inside of.

```python
for i in range(3):
    for j in range(3):
        if i == 1 and j == 1:
            break  # Only breaks the inner loop
        print(f"i={i}, j={j}")
```

If you want to break out of BOTH loops, you need a different strategy (we'll learn those techniques later in the course).

---

8. Practical Examples

Example 1: Multiplication Table

```
number = int(input("Enter a number: "))

for i in range(1, 11):
    result = number * i
    print(f"{number} x {i} = {result}")
```

Sample output for number = 5:

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
...
5 x 10 = 50
```

Example 2: Sum of Numbers

```
n = int(input("Sum numbers from 1 to: "))
total = 0

for i in range(1, n + 1):
    total = total + i

print(f"Sum of 1 to {n} is: {total}")
```

Example 3: Find First Even Number

```
numbers = [3, 7, 5, 8, 9, 12, 15]

for num in numbers:
    if num % 2 == 0:
        print(f"First even number: {num}")
        break
else:
    print("No even number found")
```

Note: The `else` on a for loop runs if the loop completes without breaking!

Example 4: Password Validator with Attempts

```
max_attempts = 3
attempts = 0

while attempts < max_attempts:
    password = input("Enter password: ")
    attempts = attempts + 1

    if password == "secret123":
        print("Access granted!")
        break
    else:
        remaining = max_attempts - attempts
        if remaining > 0:
            print(f"Wrong password. {remaining} attempts remaining.")
        else:
            print("Too many failed attempts. Access denied.")
```

Example 5: Number Guessing Game

```
secret_number = 7
guesses = 0

while True:
    guess = int(input("Guess the number (1-10): "))
    guesses = guesses + 1

    if guess == secret_number:
        print(f"Correct! You guessed it in {guesses} tries!")
        break
    elif guess < secret_number:
        print("Too low!")
    else:
        print("Too high!")
```

Example 6: Skip Negative Numbers

```
print("Enter 5 numbers (negatives will be skipped):")
count = 0

for i in range(5):
    num = int(input(f"Number {i + 1}: "))

    if num < 0:
        print("Skipping negative number")
```

```
        continue

    count = count + 1
    print(f"Added: {num}")

print(f"Total positive numbers entered: {count}")
```

Example 7: Print Pattern

```
size = int(input("Enter size: "))

for i in range(size):
    for j in range(i + 1):
        print("*", end="")
    print()  # New line after each row
```

Output for size = 5:

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

---

9. Common Patterns and Tips

Pattern 1: Counting Loop

```
for i in range(10):
    print(i)
```

Pattern 2: Repeat N Times (Don't Care About Counter)

```
for _ in range(5):
    print("Hello")
```

The underscore `_` is a convention meaning "I don't need this variable."

Pattern 3: Accumulator Pattern

```
total = 0
for i in range(1, 11):
    total = total + i
print(total)
```

Pattern 4: Input Validation

```
while True:
    age = int(input("Enter age (1-120): "))
    if 1 <= age <= 120:
        break
    print("Invalid age, try again")
```

Pattern 5: Menu System

```
while True:
    print("1. Option A")
    print("2. Option B")
    print("3. Quit")
    choice = input("Choose: ")

    if choice == "3":
        break
    # Handle other options
```

Key Takeaways

✓ **For loops:** Use when you know how many iterations (predetermined)

✓ **While loops:** Use when iterations depend on a condition (undetermined)

✓ **Range:** Creates a sequence of numbers

- `range(stop)` starts at 0
- `range(start, stop)` custom start
- `range(start, stop, step)` custom increment
- Stop value is NOT included

✓ **Break:** Exit loop immediately

✓ **Continue:** Skip to next iteration

✓ **While True:** Infinite loop, MUST have break to exit

✓ **Nested loops:** Break/continue only affect the innermost loop

✓ **Loop variable:** Temporarily holds the current value (i, count, num, etc.)

Reflection Questions

1. When should you use a for loop vs a while loop?
2. What does `range(5)` generate?
3. What does `range(2, 8)` generate?
4. Why is the stop value not included in range?
5. What's the difference between break and continue?
6. Why must you have a break in a `while True` loop?
7. In nested loops, which loop does break exit?
8. How do you count from 10 down to 1 using range?

Practice Exercises

1. **Count Down:** Print numbers from 10 to 1 using a for loop
2. **Even Numbers:** Print all even numbers from 0 to 20
3. **Factorial:** Calculate factorial of a number (n! = n × (n-1) × ... × 1)
4. **Sum Until Zero:** Keep asking for numbers and sum them until user enters 0
5. **FizzBuzz:** For numbers 1-100, print "Fizz" if divisible by 3, "Buzz" if divisible by 5, "FizzBuzz" if both
6. **Find Divisors:** Ask for a number and print all its divisors
7. **Prime Checker:** Check if a number is prime using a for loop
8. **Menu Calculator:** Create a calculator with +, -, ×, ÷ that runs until user quits
9. **Pattern Builder:** Create a pyramid pattern with asterisks
10. **Average Calculator:** Keep asking for scores until user enters -1, then calculate average