Learn Python Programming from Scratch

Topic: Loop Control Statements in Python

1. What are Loop Control Statements?

Loop control statements change the normal execution flow of loops. Python provides three main control statements: `break`, `continue`, and `pass`. Think of them as traffic signals for your loops - they help you control when to stop, skip, or pause the loop execution based on specific conditions.

2. The break Statement

The `break` statement immediately exits the current loop and continues with the next statement after the loop.

In [1]:
```
# Using break to exit a loop early
for i in range(1, 11):
    if i == 6:
        print("Breaking at 6!")
        break
    print(i)

print("Loop ended!")
```

```
1
2
3
4
5
Breaking at 6!
Loop ended!
```

3. The continue Statement

The `continue` statement skips the rest of the current iteration and jumps to the next iteration of the loop.

In [2]:
```
# Using continue to skip specific iterations
for i in range(1, 11):
    if i % 2 == 0:  # Skip even numbers
        continue
    print(f"Odd number: {i}")

print("Loop completed!")
```

```
Odd number: 1
Odd number: 3
Odd number: 5
Odd number: 7
Odd number: 9
Loop completed!
```

4. The pass Statement

The `pass` statement is a null operation - it does nothing when executed. It's used as a placeholder in syntactically required locations.

In [3]:
```python
# Using pass as placeholder
for i in range(1, 6):
    if i < 3:
        pass  # TODO: Add special handling later
    else:
        print(f"Processing number: {i}")
```

```
Processing number: 3
Processing number: 4
Processing number: 5
```

5. Using else with Loops

Python allows an `else` clause with loops. The else block executes only if the loop completes normally (not interrupted by break).

In [4]:
```python
# else clause with for loop
numbers = [2, 4, 6, 8, 10]
target = 7

for num in numbers:
    if num == target:
        print(f"Found {target}!")
        break
else:
    print(f"{target} not found in the list")
```

```
7 not found in the list
```

6. Nested Loops with Control Statements

When using control statements in nested loops, they only affect the innermost loop unless you use special techniques.

In [5]:
```python
# break and continue in nested loops
for i in range(1, 4):
    print(f"Outer loop: {i}")
    for j in range(1, 4):
        if j == 2:
            continue  # Skip j=2, continue inner loop
        if i == 2 and j == 3:
            break  # Exit inner loop when i=2 and j=3
        print(f"  Inner loop: {j}")
```

```
Outer loop: 1
  Inner loop: 1
  Inner loop: 3
Outer loop: 2
  Inner loop: 1
Outer loop: 3
  Inner loop: 1
  Inner loop: 3
```

Exercises

1. Write a program that prints numbers 1-20 but skips multiples of 3.
2. Create a search function that finds a number in a list and breaks when found.
3. Build a menu system that continues until user enters 'quit'.
4. Write a program that validates passwords until a strong one is entered.
5. Create a number guessing game using break when correct guess is made.

Practical Examples

Let's explore some practical examples of working with loop control statements in Python. These examples demonstrate real-world applications of controlling loop flow.

Data Processing and Validation System

Here's a practical example of using loop control statements to process and validate data from multiple sources.

In [6]:
```python
# Data processing system with loop control statements

# Sample student records with some invalid data
student_records = [
    {"name": "Alice", "age": 20, "grade": 85, "email": "alice@email.com"},
    {"name": "", "age": 22, "grade": 92, "email": "invalid_email"},  # Invalid nam
    {"name": "Bob", "age": -5, "grade": 78, "email": "bob@email.com"},  # Invalid
    {"name": "Charlie", "age": 21, "grade": 150, "email": "charlie@email.com"},  #
    {"name": "Diana", "age": 19, "grade": 88, "email": "diana@email.com"},
    {"name": "Eve", "age": 23, "grade": 95, "email": "eve@email.com"}
]

print("Student Data Processing System")
print("=" * 35)

valid_students = []
total_processed = 0
skipped_count = 0

for i, student in enumerate(student_records):
    total_processed += 1
    student_num = i + 1

    print(f"\nProcessing Student #{student_num}: {student.get('name', 'Unknown')}"
```

```
    # Check for missing or invalid name
    if not student.get("name") or len(student["name"].strip()) == 0:
        print("❌ Skipping: Invalid or missing name")
        skipped_count += 1
        continue

    # Check for invalid age
    age = student.get("age", 0)
    if age < 0 or age > 100:
        print(f"❌ Skipping: Invalid age ({age})")
        skipped_count += 1
        continue

    # Check for invalid grade
    grade = student.get("grade", 0)
    if grade < 0 or grade > 100:
        print(f"❌ Skipping: Invalid grade ({grade})")
        skipped_count += 1
        continue

    # Check for invalid email format (basic check)
    email = student.get("email", "")
    if "@" not in email or "." not in email:
        print(f"❌ Skipping: Invalid email ({email})")
        skipped_count += 1
        continue

    # If we reach here, student data is valid
    print("Valid student record - Added to database")
    valid_students.append(student)

    # Example: Stop processing if we have enough valid students
    if len(valid_students) >= 4:
        print(f"\nReached target of 4 valid students. Stopping processing.")
        break

# Summary statistics
print(f"\nProcessing Summary:")
print(f"Total records processed: {total_processed}")
print(f"Valid students: {len(valid_students)}")
print(f"Skipped records: {skipped_count}")
print(f"Success rate: {(len(valid_students)/total_processed)*100:.1f}%")

# Display valid students
if valid_students:
    print(f"\nValid Student Database:")
    for i, student in enumerate(valid_students, 1):
        print(f"{i}. {student['name']} (Age: {student['age']}, Grade: {student['gr
else:
    print("\nNo valid students found!")
```

```
Student Data Processing System
==================================

Processing Student #1: Alice
Valid student record - Added to database

Processing Student #2:
❌ Skipping: Invalid or missing name

Processing Student #3: Bob
❌ Skipping: Invalid age (-5)

Processing Student #4: Charlie
❌ Skipping: Invalid grade (150)

Processing Student #5: Diana
Valid student record - Added to database

Processing Student #6: Eve
Valid student record - Added to database

Processing Summary:
Total records processed: 6
Valid students: 3
Skipped records: 3
Success rate: 50.0%

Valid Student Database:
1. Alice (Age: 20, Grade: 85)
2. Diana (Age: 19, Grade: 88)
3. Eve (Age: 23, Grade: 95)
```

Interactive Menu System with Advanced Control

This example demonstrates a comprehensive menu system using various loop control statements for different scenarios.

In [7]:
```python
# Advanced interactive menu system with loop control

import random

# System data
user_account = {
    "balance": 1000.0,
    "transactions": [],
    "login_attempts": 0,
    "max_attempts": 3
}

def display_menu():
    print("\nBanking System Menu")
    print("=" * 22)
    print("1. Check Balance")
    print("2. Make Deposit")
    print("3. Make Withdrawal")
    print("4. View Transactions")
    print("5. Transfer Money")
    print("6. Account Settings")
```

```
    print("7. Exit")

print("Secure Banking System")
print("=" * 24)

# Login verification with break
while user_account["login_attempts"] < user_account["max_attempts"]:
    pin = input("Enter your 4-digit PIN: ")

    if pin == "1234":  # Simulated correct PIN
        print("Login successful!")
        user_account["login_attempts"] = 0  # Reset attempts
        break
    else:
        user_account["login_attempts"] += 1
        remaining = user_account["max_attempts"] - user_account["login_attempts"]

        if remaining > 0:
            print(f"Incorrect PIN. {remaining} attempts remaining.")
        else:
            print("Account locked due to too many failed attempts!")
            break
else:
    print("Access denied. Please contact customer service.")

# Main banking operations (only if login successful)
if user_account["login_attempts"] == 0:  # Login was successful

    while True:  # Main menu loop
        display_menu()

        try:
            choice = input("\nSelect option (1-7): ").strip()

            if choice == "1":
                # Check Balance
                print(f"Current Balance: ${user_account['balance']:.2f}")

            elif choice == "2":
                # Make Deposit
                try:
                    amount = float(input("Enter deposit amount: $"))
                    if amount <= 0:
                        print("❌ Amount must be positive!")
                        continue

                    user_account['balance'] += amount
                    user_account['transactions'].append(f"Deposit: +${amount:.2f}"
                    print(f"Deposited ${amount:.2f}. New balance: ${user_account['

                except ValueError:
                    print("❌ Please enter a valid amount!")
                    continue

            elif choice == "3":
                # Make Withdrawal
                try:
                    amount = float(input("Enter withdrawal amount: $"))
                    if amount <= 0:
                        print("Amount must be positive!")
```

```
                    continue

                if amount > user_account['balance']:
                    print("Insufficient funds!")
                    continue

                user_account['balance'] -= amount
                user_account['transactions'].append(f"Withdrawal: -${amount:.2
                print(f"Withdrew ${amount:.2f}. New balance: ${user_account['b

            except ValueError:
                print("Please enter a valid amount!")
                continue

        elif choice == "4":
            # View Transactions
            if user_account['transactions']:
                print("\nRecent Transactions:")
                for i, transaction in enumerate(user_account['transactions'][-
                    print(f"{i}. {transaction}")
            else:
                print("No transactions found.")

        elif choice == "5":
            # Transfer Money (Simulated)
            print("Money Transfer Service")

            # Check if sufficient balance for any transfer
            if user_account['balance'] < 10:
                print("Insufficient balance for transfer (minimum $10)")
                continue

            recipient = input("Enter recipient account number: ")

            # Skip if invalid account format
            if len(recipient) != 10 or not recipient.isdigit():
                print("Invalid account number format!")
                continue

            try:
                amount = float(input("Enter transfer amount: $"))
                if amount < 10:
                    print("Minimum transfer amount is $10!")
                    continue

                if amount > user_account['balance']:
                    print("Insufficient funds!")
                    continue

                # Simulate transfer processing
                print("Processing transfer...")
                user_account['balance'] -= amount
                user_account['transactions'].append(f"Transfer to {recipient}:
                print(f"Transfer successful! New balance: ${user_account['bala

            except ValueError:
                print("Please enter a valid amount!")
                continue

        elif choice == "6":
```

```
                # Account Settings (placeholder)
                print("Account Settings")
                print("This feature is under development.")
                pass  # Placeholder for future implementation

            elif choice == "7":
                # Exit
                print("Thank you for banking with us!")
                print("Logging out securely...")
                break

            else:
                print("Invalid option! Please select 1-7.")
                continue

        except KeyboardInterrupt:
            print("\n\nSession interrupted by user.")
            print("Goodbye!")
            break
        except Exception as e:
            print(f"An error occurred: {e}")
            continue

print("\nSession ended.")
```

```
Secure Banking System
=======================
Login successful!

Banking System Menu
====================
1. Check Balance
2. Make Deposit
3. Make Withdrawal
4. View Transactions
5. Transfer Money
6. Account Settings
7. Exit
Deposited $2000.00. New balance: $3000.00

Banking System Menu
====================
1. Check Balance
2. Make Deposit
3. Make Withdrawal
4. View Transactions
5. Transfer Money
6. Account Settings
7. Exit

Recent Transactions:
1. Deposit: +$2000.00

Banking System Menu
====================
1. Check Balance
2. Make Deposit
3. Make Withdrawal
4. View Transactions
5. Transfer Money
6. Account Settings
7. Exit
Account Settings
This feature is under development.

Banking System Menu
====================
1. Check Balance
2. Make Deposit
3. Make Withdrawal
4. View Transactions
5. Transfer Money
6. Account Settings
7. Exit
Thank you for banking with us!
Logging out securely...

Session ended.
```

Key Loop Control Rules to Remember

Let's review the important rules and best practices for working with loop control statements:

- Use `break` to exit loops early when a specific condition is met
- Use `continue` to skip the current iteration and move to the next one
- Use `pass` as a placeholder for code that will be implemented later
- The `else` clause in loops executes only if the loop completes without break
- In nested loops, break and continue only affect the innermost loop
- Always consider the logic flow when using control statements
- Use meaningful conditions that make the code's intent clear
- Test your control statements with different data to ensure proper behavior
- Avoid excessive use of break and continue as it can make code hard to follow
- Use control statements to handle edge cases and error conditions gracefully
- Document complex control flow with comments for future maintenance
- Consider refactoring complex control logic into separate functions

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

# Example 1: Data validation with comprehensive error handling
print("Survey Data Collection")
print("=" * 25)

survey_responses = []
max_responses = 5
current_response = 0

while current_response < max_responses:
    print(f"\nResponse {current_response + 1} of {max_responses}")

    try:
        # Get age input
        age = input("Enter age (or 'skip' to skip this response): ").strip()

        if age.lower() == 'skip':
            print("Skipping this response...")
            current_response += 1
            continue

        age = int(age)

        # Validate age range
        if age < 13 or age > 100:
            print("Age must be between 13 and 100!")
            continue  # Don't increment counter, try again

        # Get satisfaction rating
        rating = input("Rate satisfaction (1-5): ").strip()

        if not rating.isdigit() or int(rating) not in range(1, 6):
            print("Rating must be between 1 and 5!")
            continue  # Don't increment counter, try again

        rating = int(rating)

        # Get optional feedback
        feedback = input("Optional feedback (or press Enter to skip): ").strip()

        # Store valid response
        response = {
```

```
            "age": age,
            "rating": rating,
            "feedback": feedback if feedback else "No feedback"
        }

        survey_responses.append(response)
        current_response += 1

        print("Response recorded successfully!")

        # Early exit option
        if current_response < max_responses:
            continue_survey = input("Continue with survey? (y/n): ").strip().lower
            if continue_survey in ['n', 'no']:
                print("Survey completed early.")
                break

    except ValueError:
        print("Please enter valid numeric values!")
        continue
    except KeyboardInterrupt:
        print("\n\nSurvey cancelled by user.")
        break

# Process results
if survey_responses:
    print(f"\nSurvey Results ({len(survey_responses)} responses):")

    total_rating = 0
    age_groups = {"13-18": 0, "19-30": 0, "31-50": 0, "51+": 0}

    for i, response in enumerate(survey_responses, 1):
        print(f"{i}. Age: {response['age']}, Rating: {response['rating']}/5")

        total_rating += response['rating']

        # Categorize age groups
        age = response['age']
        if age <= 18:
            age_groups["13-18"] += 1
        elif age <= 30:
            age_groups["19-30"] += 1
        elif age <= 50:
            age_groups["31-50"] += 1
        else:
            age_groups["51+"] += 1

    avg_rating = total_rating / len(survey_responses)
    print(f"\nAverage Rating: {avg_rating:.1f}/5")
    print(f"Age Group Distribution: {age_groups}")

else:
    print("\nNo survey responses collected.")
```

```
Survey Data Collection
======================

Response 1 of 5
Response recorded successfully!
Survey completed early.

Survey Results (1 responses):
1. Age: 42, Rating: 4/5

Average Rating: 4.0/5
Age Group Distribution: {'13-18': 0, '19-30': 0, '31-50': 1, '51+': 0}
```

Course Information

Learn Python Programming from Scratch

Author: Prakash Ukhalkar

Topic: Python Control Flow - Loop Control Statements

Built with ❤️ for the Python learning community