Python Lab Manual 04: Iteration and Loops (Final Edition)

Objective

To practice and verify fundamental loop structures in Python by following detailed, line-by-line instructions, enhancing understanding of **while**, **for**, **break**, and **continue**.

---

1. The while Loop

Task 1.1: Simple Accumulator (Ask for positive number)

This task uses a **while loop** to demonstrate **input validation**, forcing the program to repeatedly ask the user for input until a specific condition (entering a positive number) is met.

- **Guidance:** Initialize the control variable to a value that ensures the loop runs first.
    - **Code Line:** num = -1
    - **Explanation:** Sets the initial state to satisfy the loop condition.
- **Guidance:** Instruct the user on what to enter.
    - **Code Line:** print("Please enter a positive integer to continue.")
- **Guidance:** Start the **while loop**, checking for zero or negative values.
    - **Code Line:** while num <= 0:
    - **Explanation:** Loop continues as long as num is not positive.
- **Guidance:** Prompt the user for input and update the variable inside the loop.
    - **Code Line:** num = int(input("Enter a number: "))
    - **Explanation:** Updates num; if positive, the loop condition fails and the loop ends.
- **Guidance:** Print the success message outside the loop.
    - **Code Line:** print(f"Thank you! You entered the positive number: {num}")

Complete Program (for Verification):

Python

```
num = -1
print("Please enter a positive integer to continue.")
```

```
while num <= 0:
    num = int(input("Enter a number: "))

print(f"Thank you! You entered the positive number: {num}")
```

Expected Output (Example Interaction):

```
Please enter a positive integer to continue.
Enter a number: -10
Enter a number: 0
Enter a number: 42
Thank you! You entered the positive number: 42
```

Task 1.2: Password Retry Limit (Max 3 attempts)

This exercise demonstrates using a **while loop** with an auxiliary counter to implement a **limited retry mechanism**. The loop must also use **break** to exit early on success.

- **Guidance:** Define the correct password and initialize the attempt counter.
    - **Code Line:** SECRET = "Python3"
    - **Code Line:** attempts = 0
- **Guidance:** Set the **while loop** condition to run while attempts are strictly less than 3.
    - **Code Line:** while attempts < 3:
- **Guidance:** Get the user's guess, showing the attempt count.
    - **Code Line:** guess = input(f"Attempt {attempts + 1}/3. Enter password: ")
- **Guidance:** Check the guess. If correct, print success and stop the loop immediately.
    - **Code Line:** if guess == SECRET:
    - **Code Line:** print("Login Successful!")
    - **Code Line:** break
- **Guidance:** If the guess was wrong, increment the counter and notify the user.
    - **Code Line:** attempts += 1
    - **Code Line:** print("Incorrect password.")
- **Guidance:** Check outside the loop if the limit was reached to print the final failure message.
    - **Code Line:** if attempts == 3:
    - **Code Line:** print("Access Denied. Max attempts reached.")

Complete Program (for Verification):

Python

```
SECRET = "Python3"
attempts = 0

while attempts < 3:
    guess = input(f"Attempt {attempts + 1}/3. Enter password: ")

    if guess == SECRET:
        print("Login Successful!")
        break

    attempts += 1
    print("Incorrect password.")

if attempts == 3:
    print("Access Denied. Max attempts reached.")
```

Expected Output (Failure):

```
Attempt 1/3. Enter password: wrong
Incorrect password.
Attempt 2/3. Enter password: guess
Incorrect password.
Attempt 3/3. Enter password: last
Incorrect password.
Access Denied. Max attempts reached.
```

---

2. Using for Loop with Lists and range()

Task 2.3: List of Multiples (Create list: 7, 14, 21, 28, 35)

This task uses a **for loop** over a simple **range()** to dynamically generate and populate a list with calculated values (the first five multiples of 7).

- **Guidance:** Initialize an empty list.
    - **Code Line:** multiples_of_7 = []
- **Guidance:** Start a **for loop** to generate the factors 1 through 5.
    - **Code Line:** for i in range(1, 6):
    - **Explanation:** i will be 1, 2, 3, 4, 5.
- **Guidance:** Calculate the current multiple.
    - **Code Line:** multiple = 7 * i
- **Guidance:** Add the calculated value to the list using the .append() method.
    - **Code Line:** multiples_of_7.append(multiple)
- **Guidance:** Print the final list.

- **Code Line:** print("Multiples of 7:", multiples_of_7)

Complete Program (for Verification):

Python

```
multiples_of_7 = []

for i in range(1, 6):
    multiple = 7 * i
    multiples_of_7.append(multiple)

print("Multiples of 7:", multiples_of_7)
```

Expected Output:

```
Multiples of 7: [7, 14, 21, 28, 35]
```

Task 3.2: Price Calculation (Sum of prices)

This task illustrates the power of the **for loop** for **aggregation** by iterating directly over a list of numerical values to calculate their total sum.

- **Guidance:** Define the list of prices and initialize the total accumulator to zero.
  - **Code Line:** prices = [15.50, 40.00, 5.25, 100.00]
  - **Code Line:** total = 0
- **Guidance:** Start the **for loop**, iterating directly over each price item in the list.
  - **Code Line:** for price in prices:
- **Guidance:** Accumulate the sum in the total variable.
  - **Code Line:** total += price
  - **Explanation:** Shorthand for total = total + price.
- **Guidance:** Print the final total, formatted to two decimal places.
  - **Code Line:** print(f"Total cost: ${total:.2f}")

Complete Program (for Verification):

Python

```
prices = [15.50, 40.00, 5.25, 100.00]
total = 0

for price in prices:
    total += price

print(f"Total cost: ${total:.2f}")
```

Expected Output:

```
Total cost: $160.75
```

Task 4.2: Printing Odd Numbers (10 to 30 inclusive)

This task uses the powerful three-argument form of **range(start, stop, step)** with a **for loop** to efficiently generate only the odd numbers within a specified range.

- **Guidance:** Print a header for clarity.
    - **Code Line:** print("Odd numbers between 10 and 30:")
- **Guidance:** Use **range()** to start at 11, stop at 31, with a step of 2.
    - **Code Line:** for i in range(11, 31, 2):
    - **Explanation:** Starts at the first odd number (11), and a step of 2 skips all even numbers.
- **Guidance:** Print the current number generated by range().
    - **Code Line:** print(i)

Complete Program (for Verification):

Python

```
print("Odd numbers between 10 and 30:")
for i in range(11, 31, 2):
    print(i)
```

Expected Output:

```
Odd numbers between 10 and 30:
11
13
15
17
19
21
23
25
27
29
```

Task 4.3: Indexing a List (Print index and value)

This task demonstrates how to use the **len()** function with **range()** in a **for loop** to access both the **index** and the **value** of items in a list.

- **Guidance:** Define the list of data.
    - **Code Line:** planets = ["Mercury", "Venus", "Earth", "Mars"]
- **Guidance:** Use range(len()) to loop over the valid **indices** (0, 1, 2, 3).
    - **Code Line:** for i in range(len(planets)):
- **Guidance:** Access the planet name using the current index i.
    - **Code Line:** planet = planets[i]

    - **Explanation:** Uses bracket notation to retrieve the value corresponding to the index.
- **Guidance:** Print the formatted output.
    - **Code Line:** print(f"Planet at Index {i}: {planet}")

Complete Program (for Verification):

Python

```
planets = ["Mercury", "Venus", "Earth", "Mars"]

for i in range(len(planets)):
    planet = planets[i]
    print(f"Planet at Index {i}: {planet}")
```

Expected Output:

```
Planet at Index 0: Mercury
Planet at Index 1: Venus
Planet at Index 2: Earth
Planet at Index 3: Mars
```

---

3. Loop Control Keywords (continue and break)

Task 5.1: Skipping a Multiple (continue) (Skip multiples of 3)

This task focuses on the **continue** keyword, which is used to **skip the remaining code** in the current loop iteration and immediately move to the next item or step.

- **Guidance:** Start the loop for numbers 1 through 15.
    - **Code Line:** for i in range(1, 16):
- **Guidance:** Check if the number is divisible by 3 using the **modulo operator** (%).
    - **Code Line:** if i % 3 == 0:
- **Guidance:** If divisible, skip the rest of this iteration.
    - **Code Line:** continue
    - **Explanation:** Jumps directly to the next number, bypassing the print statement below.
- **Guidance:** Print the number on the same line if it was not skipped.
    - **Code Line:** print(i, end=" ")
    - **Code Line:** print() # Final newline

Complete Program (for Verification):

Python

```
print("Numbers not divisible by 3 (1-15):")
```

```
for i in range(1, 16):
    if i % 3 == 0:
        continue
    print(i, end=" ")
print()
```

Expected Output:

```
Numbers not divisible by 3 (1-15):
1 2 4 5 7 8 10 11 13 14
```

Task 6.1: Search and Stop (break) (Search for "P-99")

This task focuses on the **break** keyword, which is used to **immediately terminate the entire loop** once a target item is found, saving processing time.

- **Guidance:** Define the data and the target.
  - **Code Line:** codes = ["P-10", "P-55", "P-99", "P-101", "P-150"]
  - **Code Line:** target = "P-99"
- **Guidance:** Loop through the codes.
  - **Code Line:** for code in codes:
- **Guidance:** Print the current item being checked.
  - **Code Line:** print(f"Checking {code}...")
- **Guidance:** Check for the target code.
  - **Code Line:** if code == target:
- **Guidance:** If found, print success and use **break** to stop the loop.
  - **Code Line:** print(f"Target {target} found! Stopping search.")
  - **Code Line:** break
  - **Explanation:** The loop terminates here, and subsequent codes (P-101, P-150) are not checked.

Complete Program (for Verification):

Python

```
codes = ["P-10", "P-55", "P-99", "P-101", "P-150"]
target = "P-99"

for code in codes:
    print(f"Checking {code}...")
    if code == target:
        print(f"Target {target} found! Stopping search.")
        break
```

Expected Output:

```
Checking P-10...
Checking P-55...
Checking P-99...
```

Target P-99 found! Stopping search.

Python Lab Manual 04: Iteration and Loops

Objective

To understand, implement, and verify **while loops** and **for loops** using full, runnable Python code, focusing on structured guidance and basic control flow.

---

1. The while Loop

The **while loop** repeats a code block as long as its condition is **True**.

Task 1.1: Simple Accumulator

Ask the user for input repeatedly until a positive integer is entered (assume valid integer input for simplicity).

- **Guidance:**
    - Initialize the control variable num to a negative value (e.g., -1) so the while loop condition is met initially.
    - Set the while loop condition to run as long as num <= 0.
    - Inside the loop, prompt the user for input and convert it to an integer.
    - Print a success message outside the loop.
- **Program:**

  Python

  ```
  # Task 1.1: Simple Accumulator
  num = -1
  print("Please enter a positive integer to continue.")

  while num <= 0:
      # Assuming the user enters a valid integer
      num = int(input("Enter a number: "))

  print(f"Thank you! You entered the positive number: {num}")
  ```
- **Expected Output (Example Interaction):**

  ```
  Please enter a positive integer to continue.
  Enter a number: -10
  Enter a number: 0
  Enter a number: 50
  Thank you! You entered the positive number: 50
  ```

Task 1.2: Password Retry Limit

Allow a maximum of **3 attempts** to enter the correct password ("Python3").

- **Guidance:**
    - Define the constant SECRET = "Python3" and initialize attempts = 0.
    - Use a while loop condition: attempts < 3.
    - Inside the loop, get the user's guess.
    - Use an if statement to check the guess. If correct, print success and use **break**.
    - Increment the attempts counter in all cases after checking.
    - Outside the loop, check if attempts == 3 to print the "Access Denied" message.
- **Program:**

  Python

```
# Task 1.2: Password Retry Limit
SECRET = "Python3"
attempts = 0

while attempts < 3:
    guess = input(f"Attempt {attempts + 1}/3. Enter password: ")

    if guess == SECRET:
        print("Login Successful!")
        break

    attempts += 1
    print("Incorrect password.")

if attempts == 3:
    print("Access Denied. Max attempts reached.")
```

- **Expected Output (Failure):**

```
Attempt 1/3. Enter password: 123
Incorrect password.
Attempt 2/3. Enter password: abc
Incorrect password.
Attempt 3/3. Enter password: xyz
Incorrect password.
Access Denied. Max attempts reached.
```

Task 1.3: Number Halver

Start at 100 and repeatedly divide by 2, printing the result, until the value is less than 5.

- **Guidance:**
    - Initialize n = 100.
    - Set the while loop condition to run as long as n >= 5.

    - Inside the loop, print the current value of n (using :.2f for formatting).
    - Update the control variable: n = n / 2.
- **Program:**

  Python

  ```
  # Task 1.3: Number Halver
  n = 100
  print(f"Starting value: {n}")

  while n >= 5:
      print(f"Current n: {n:.2f}")
      n = n / 2

  print(f"Final value ({n:.2f}) is now less than 5.")
  ```
- **Expected Output:**

  ```
  Starting value: 100
  Current n: 100.00
  Current n: 50.00
  Current n: 25.00
  Current n: 12.50
  Current n: 6.25
  Final value (3.12) is now less than 5.
  ```

---

2. Creating Lists of Values

Lists are the primary sequence type for iteration in Python.

Task 2.1: Simple List Creation

Create a list containing mixed data types (strings, integers, boolean) and print its contents and length.

- **Guidance:**
    - Create a list using square brackets [] and include at least two different data types.
    - Use the built-in print() function to display the list.
    - Use the built-in len() function to get the list's size.
- **Program:**

  Python

  ```
  # Task 2.1: Simple List Creation
  mixed_list = ["Lahore", "Karachi", 7, 14, True]

  print("The List:", mixed_list)
  print(f"List length: {len(mixed_list)}")
  ```
- **Expected Output:**

```
The List: ['Lahore', 'Karachi', 7, 14, True]
List length: 5
```

Task 2.2: Dynamic List Population

Use a for loop to collect 5 names from the user, adding each to an initially empty list.

- **Guidance:**
  - Initialize an empty list called names.
  - Use a for loop with range(5) to repeat the input process 5 times.
  - Inside the loop, use input() to get a name.
  - Use the .append() list method to add the new name to the list.
- **Program:**

  Python

```
# Task 2.2: Dynamic List Population
names = []

print("Enter 5 names:")
for i in range(5):
    new_name = input(f"Enter name #{i+1}: ")
    names.append(new_name)

print("\nCollected Names:", names)
```

- **Expected Output (Example Interaction):**

```
Enter 5 names:
Enter name #1: Ali
Enter name #2: Sara
Enter name #3: Usman
Enter name #4: Ayesha
Enter name #5: Zaki

Collected Names: ['Ali', 'Sara', 'Usman', 'Ayesha', 'Zaki']
```

Task 2.3: List of Multiples

Create a list containing the first 5 positive multiples of 7 (i.e., 7, 14, 21, 28, 35).

- **Guidance:**
  - Initialize an empty list multiples_of_7.
  - Loop using range(1, 6) to get the factors 1 through 5.
  - Inside the loop, calculate the multiple (7 * i) and append it to the list.
- **Program:**

  Python

```
# Task 2.3: List of Multiples
multiples_of_7 = []
```

```
for i in range(1, 6):
    multiple = 7 * i
    multiples_of_7.append(multiple)

print("Multiples of 7:", multiples_of_7)
```

- **Expected Output:**

  Multiples of 7: [7, 14, 21, 28, 35]

---

3. Using for Loop with List

The **for loop** is used to process every element in a sequence.

Task 3.1: Item Printer

Print each city name and its length from a predefined list.

- **Guidance:**
  - Define the list cities.
  - Use a for loop to iterate directly over the list items (for city in cities:).
  - Inside the loop, use the len() function to find the length of the current city string.
- **Program:**

  Python

```
# Task 3.1: Item Printer
cities = ["Lahore", "Quetta", "Peshawar", "Karachi"]

print("City Data:")
for city in cities:
    length = len(city)
    print(f"- {city} has {length} letters.")
```

- **Expected Output:**

```
City Data:
- Lahore has 6 letters.
- Quetta has 6 letters.
- Peshawar has 8 letters.
- Karachi has 7 letters.
```

Task 3.2: Price Calculation

Calculate the **total sum** of prices in a list.

- **Guidance:**
  - Initialize an accumulator variable total = 0 before the loop.
  - Loop through the prices list.
  - Inside the loop, update the accumulator: total += price.

- **Program:**

  Python

  ```
  # Task 3.2: Price Calculation
  prices = [15.50, 40.00, 5.25, 100.00]
  total = 0

  for price in prices:
      total += price

  print(f"Total cost: ${total:.2f}")
  ```

- **Expected Output:**

  ```
  Total cost: $160.75
  ```

---

4. Using for Loop with range()

The range() function generates an integer sequence for counting.

Task 4.1: Counting with range()

Print the numbers from 5 down to 1 using range(start, stop, step).

- **Guidance:**
  - Use the three-argument form of range().
  - Start at **5**.
  - The stop value must be **0** (to include 1).
  - The step must be **-1**.
- **Program:**

  Python

  ```
  # Task 4.1: Counting with range()
  print("Countdown:")
  for i in range(5, 0, -1):
      print(i)
  print("Lift Off!")
  ```

- **Expected Output:**

  ```
  Countdown:
  5
  4
  3
  2
  1
  Lift Off!
  ```

Task 4.2: Printing Odd Numbers

Print all odd numbers between 10 and 30 (inclusive).

- **Guidance:**

- Use range() to start at the first odd number in the range (**11**).
- Use a step of **2** to skip the even numbers.
- The stop value must be **31** to ensure 29 is included.

- **Program:**

  Python

  ```
  # Task 4.2: Printing Odd Numbers
  print("Odd numbers between 10 and 30:")
  for i in range(11, 31, 2):
      print(i)
  ```

- **Expected Output:**

  ```
  Odd numbers between 10 and 30:
  11
  13
  15
  17
  19
  21
  23
  25
  27
  29
  ```

Task 4.3: Indexing a List

Use range(len(my_list)) to print both the **index** and the **value** for every item in a list.

- **Guidance:**
  - Use len() to get the size of the list.
  - Use range() with the length to generate the indices (0, 1, 2...).
  - Inside the loop, use bracket notation (list[index]) to access the element.

- **Program:**

  Python

  ```
  # Task 4.3: Indexing a List
  planets = ["Mercury", "Venus", "Earth", "Mars"]

  for i in range(len(planets)):
      planet = planets[i]
      print(f"Planet at Index {i}: {planet}")
  ```

- **Expected Output:**

  ```
  Planet at Index 0: Mercury
  Planet at Index 1: Venus
  Planet at Index 2: Earth
  Planet at Index 3: Mars
  ```

---

5. Skipping Numbers in Range

The **continue** keyword skips the current loop iteration.

Task 5.1: Skipping a Multiple

Print numbers from 1 to 15, but skip any number that is a multiple of **3**.

- **Guidance:**
  - Loop through the range (1, 16).
  - Use the **modulo operator** (%) to check for divisibility: i % 3 == 0.
  - If the condition is true, use the **continue** keyword.
- **Program:**

  Python

  ```
  # Task 5.1: Skipping a Multiple
  print("Numbers not divisible by 3 (1-15):")
  for i in range(1, 16):
      if i % 3 == 0:
          continue  # Skip multiples of 3 (3, 6, 9, 12, 15)
      print(i, end=" ")
  print()
  ```
- **Expected Output:**

  ```
  Numbers not divisible by 3 (1-15):
  1 2 4 5 7 8 10 11 13 14
  ```

Task 5.2: Skipping Vowels

Iterate through a string and print only the consonants.

- **Guidance:**
  - Define a string of vowels (vowels = "aeiou").
  - Loop directly over the characters in the word.
  - Use if char in vowels: to check if the current character is a vowel.
  - If it is a vowel, use **continue**.
  - Use print(char, end="") to print characters on the same line.
- **Program:**

  Python

  ```
  # Task 5.2: Skipping Vowels
  word = "engineering"
  vowels = "aeiou"

  print(f"Consonants in '{word}': ", end="")
  for char in word:
      if char in vowels:
          continue
  ```

```
        print(char, end="")

    print()
```

- **Expected Output:**

  Consonants in 'engineering': ngnrng

---

6. Controlling Loop Termination

The **break** keyword immediately exits the entire loop.

Task 6.1: Search and Stop

Search a list for the code "P-99" and stop immediately when found.

- **Guidance:**
  - Loop through the list of codes.
  - Use an if statement to check if the current code matches the target.
  - If it matches, print a success message and use **break** to stop searching the rest of the list.
- **Program:**

  Python

```
# Task 6.1: Search and Stop
codes = ["P-10", "P-55", "P-99", "P-101", "P-150"]
target = "P-99"

for code in codes:
    print(f"Checking {code}...")
    if code == target:
        print(f"Target {target} found! Stopping search.")
        break
```

- **Expected Output:**

```
Checking P-10...
Checking P-55...
Checking P-99...
Target P-99 found! Stopping search.
```

Task 6.2: Input Validation with while and break

Use a while True loop to force the user to enter an age between 18 and 60.

- **Guidance:**
  - Start an intentional infinite loop: while True:.
  - Prompt the user for input and convert it to an integer.
  - Check the validation range (18 <= age <= 60).
  - If valid, use **break** to exit the infinite loop.
- **Program:**

Python

```
# Task 6.2: Input Validation with while and break
print("Enter age (18-60) to proceed.")
while True:
    # Assuming the user enters a valid integer
    age = int(input("Age: "))

    if 18 <= age <= 60:
        print(f"Age {age} accepted. Moving on.")
        break  # Exit loop when valid
    else:
        print("Age out of range. Must be between 18 and 60. Try again.")
```

- **Expected Output (Example Interaction):**

```
Enter age (18-60) to proceed.
Age: 17
Age out of range. Must be between 18 and 60. Try again.
Age: 65
Age out of range. Must be between 18 and 60. Try again.
Age: 25
Age 25 accepted. Moving on.
```

Task 6.3: First Negative Number

Iterate through a list of temperatures and stop the loop immediately upon encountering the first negative value.

- **Guidance:**
  - Loop through the list of temps.
  - Check the condition if t < 0.
  - If true, print an alert and use **break**.
- **Program:**

Python

```
# Task 6.3: First Negative Number
temps = [25, 18, 5, 0, -3, 10, -5]

for t in temps:
    if t < 0:
        print(f"ALERT! Freezing detected at {t}°C.")
        break
    print(f"Current temp {t}°C: OK.")
```

- **Expected Output:**

```
Current temp 25°C: OK.
Current temp 18°C: OK.
Current temp 5°C: OK.
Current temp 0°C: OK.
ALERT! Freezing detected at -3°C.
```

---

7. Challenge Tasks (Do It Yourself)

Challenge 7.1: Prime Number Checker

Determine if a user-provided integer (n) is a prime number.

- **Guidance:**
    1. Get the number n from the user (assuming $n \ge 2$).
    2. Set a boolean flag is_prime = True.
    3. Loop from $i = 2$ up to (but not including) n.
    4. Inside the loop, check if n is divisible by i (n % i == 0).
    5. If divisible, set is_prime = False and use **break**.
    6. Print the result based on the final value of the is_prime flag.
- **Program:** (Student must complete the logic)

  Python

```
# Challenge 7.1: Prime Number Checker - Complete this program
# Example Input: 17, 15
pass
```

Challenge 7.2: Text Reverser

Reverse a user-entered word using a for loop with negative steps in range().

- **Guidance:**
    1. Get the word and its length L.
    2. Initialize reversed_word = "".
    3. Use the for i in range(L - 1, -1, -1): structure to iterate indices backward.
    4. Append word[i] to reversed_word in each step.
- **Program:** (Student must complete the logic)

  Python

```
# Challenge 7.2: Text Reverser - Complete this program
# Example Input: "programming"
pass
```

Challenge 7.3: Fibonacci Sequence Generator

Generate and print the first 10 terms of the Fibonacci sequence using a while loop.

- **Guidance:**
    1. Initialize a = 0, b = 1, and count = 0.
    2. The while loop runs as long as count < 10.

3. Inside the loop:
    - Print the current term (a).
    - Calculate the next term: next_term = a + b.
    - Update the terms: a becomes b, and b becomes next_term.
    - Increment count.

- **Program:** (Student must complete the logic)

  Python

  ```
  # Challenge 7.3: Fibonacci Sequence Generator - Complete this program
  # Sequence: 0 1 1 2 3 5 8 13 21 34...
  pass
  ```