SOFT40161_Lab02

October 7, 2024

Nottingham Trent University

Department of Computer Science

1 SOFT40161 - Introduction to Computer Programming: Lab 02

Try to buid up the concepts! Don't just execute the codes without knowing it's meaning.

1.1 Lab Learning Outcomes

By the end of this lab, you will be able to: - **Understand and apply control structures:** Implement sequence, branching, and looping to control program flow. - **Utilize conditional statements:** Use if, elif, and else statements to handle decision-making in your code. - **Work with looping structures:** Implement for and while loops to automate repetitive tasks. - **Implement nested control structures:** Use nested if statements and multiple for loops for more complex scenarios. - **Identify and avoid infinite loops:** Understand infinite loops and how to prevent or break them in both for and while loops. - **Write readable and efficient code:** Practice proper code structuring and commenting to enhance readability and maintainability.

These outcomes will help you gain a strong foundation in programming, particularly in controlling the logic and flow of Python programs.

1.2 Control Structure with Python in Jupyter Notebook

- Welcome to Lab 2! In this session, we will explore one of the core concepts of programming: control structures. These structures allow us to control the flow of execution in a program, making it more flexible and dynamic. By mastering these tools, you'll gain the ability to make decisions within your code, execute tasks repeatedly, and handle different scenarios efficiently.
- In this lab, we will focus on four essential control structures:
  - Sequence: The default mode where code is executed line by line.
  - Branching: Using if-else statements to choose different paths based on conditions.

- Conditions: Checking whether certain criteria are met, allowing for complex decision-making.
- Looping: Repeating tasks using for and while loops to handle repetitive tasks.

Throughout the lab, we will work through examples with increasing complexity, including nested if statements, multiple for loops, and understanding infinite loops. You'll also have the opportunity to apply your knowledge through hands-on exercises designed to solidify your understanding.

By the end of this lab, you should feel confident in using control structures to manage how your programs run. These skills will be essential as we move forward to more advanced topics. Let's get started!

1.2.1 Sequence

In Python, code is executed line by line in the order it appears, unless altered by control structures like loops or conditionals.

[9]:
```
# Sequence Example: Simple steps to calculate average
number1 = 10
number2 = 20
number3 = 30

# Step-by-step calculation
total = number1 + number2 + number3
average = total / 3

print("Total:", total)
print("Average:", average)
```

```
Total: 60
Average: 20.0
```

Explanation Here, three numbers are assigned, and we calculate their total and average in a sequence. Python executes the lines one after the other without any conditions or loops.

[11]:
```
# Another Example: Sequential code
print("Step 1: Data loading")
data = [1, 2, 3, 4, 5]
print("Step 2: Data processing")
processed_data = [x * 2 for x in data]
print("Step 3: Data output")
print("Processed Data:", processed_data)
```

```
Step 1: Data loading
Step 2: Data processing
Step 3: Data output
Processed Data: [2, 4, 6, 8, 10]
```

Explanation Each print and processing line runs sequentially, showing how data flows step by step. This is often the most basic flow but crucial for understanding more complex constructs.

1.2.2 Branching (if, elif, else)

Branching allows your code to make decisions based on conditions. The most common form is the if-elif-else structure.

[14]:
```
# Branching Example: Checking exam grade
grade = 75

if grade >= 85:
    print("You received an A!")
elif grade >= 70:
    print("You received a B.")
elif grade >= 50:
    print("You received a C.")
else:
    print("You failed the exam.")
```

```
You received a B.
```

Explanation This example checks the grade and prints a message depending on the value. Python evaluates each condition sequentially, and only one block of code is executed.

Conditions with Multiple Comparisons You can combine conditions using logical operators like and, or, and not.

[17]:
```
# Multiple Conditions Example: Checking if a number is in a specific range
age = 19

if age >= 18 and age <= 25:
    print("You are eligible for the student discount.")
else:
    print("You are not eligible for the student discount.")
```

```
You are eligible for the student discount.
```

Explanation: The condition checks if the age falls between 18 and 25 (inclusive). Both comparisons must be true for the message to be printed.

Nested if Statements You can also use if statements within other if statements to create more complex decision-making flows.

[20]:
```
# Example: Nested if-else
temperature = 25
humidity = 40

if temperature > 30:
    if humidity > 50:
        print("It's hot and humid.")
    else:
        print("It's hot but dry.")
else:
```

```
    if humidity > 50:
        print("It's cool but humid.")
    else:
        print("It's cool and dry.")
```

```
It's cool and dry.
```

Explanation: Nested if statements allow us to check multiple conditions within different scenarios, which is useful when making decisions that depend on several factors at once, such as climate conditions or sensor data.

1.2.3 Looping: for Loops

Loops allow you to repeat a block of code multiple times. The for loop is ideal when you know the number of iterations in advance.

[23]:
```
# For Loop Example: Calculate the sum of numbers from 1 to 100
total_sum = 0

for number in range(1, 101):
    total_sum += number  # Add each number to total_sum

print("The sum of numbers from 1 to 100 is:", total_sum)
```

```
The sum of numbers from 1 to 100 is: 5050
```

Explanation: The loop iterates through numbers from 1 to 100, accumulating the total sum. This loop has 100 iterations, each adding a new number to total_sum.

[25]:
```
# Example2: Simple for loop
data = [1, 2, 3, 4, 5]
for value in data:
    print("Current value:", value)
```

```
Current value: 1
Current value: 2
Current value: 3
Current value: 4
Current value: 5
```

Explanation: The loop iterates over each element of the list data, printing the current value. for loops are often used for iterating over datasets, processing each element one by one.

Looping: while Loops The while loop is used when the number of iterations is not known in advance, and it keeps running as long as a condition is true.

[28]:
```
# While Loop Example: Countdown from 10 to 1
count = 10

while count > 0:
```

```
    print("Countdown:", count)
    count -= 1  # Decrease count by 1

print("Blast off!")
```

```
Countdown: 10
Countdown: 9
Countdown: 8
Countdown: 7
Countdown: 6
Countdown: 5
Countdown: 4
Countdown: 3
Countdown: 2
Countdown: 1
Blast off!
```

- write multiple statements separated by semi colon(s) in one line as follows

Explanation: This loop continues to execute as long as count is greater than zero. Once the condition becomes false, the loop stops, and "Blast off!" is printed.

Multiple for Loops You can nest for loops within each other to handle multidimensional data.

[32]:
```
# Example: Multiple for loops (Nested)
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
for row in matrix:
    for value in row:
        print(value, end=" ")
    print()  # To create a new line after each row
```

```
1 2 3
4 5 6
7 8 9
```

Explanation: The outer loop iterates over each row of the matrix, and the inner loop iterates over each element in that row. Nested for loops are particularly useful when working with two-dimensional data such as matrices, grids, or tables.

Infinite for Loop Example An infinite loop runs forever unless manually interrupted. In a for loop, this can happen if you use an iterator that never ends.

[35]:
```
# Example: Infinite while loop
count = 0
while True:
    print("This will run forever unless you stop it! Iteration no = ",count)
    if count == 10:  # Breaking condition to avoid infinite loop
        break
    count += 1
```

```
This will run forever unless you stop it! Iteration no =  0
This will run forever unless you stop it! Iteration no =  1
This will run forever unless you stop it! Iteration no =  2
This will run forever unless you stop it! Iteration no =  3
This will run forever unless you stop it! Iteration no =  4
This will run forever unless you stop it! Iteration no =  5
This will run forever unless you stop it! Iteration no =  6
This will run forever unless you stop it! Iteration no =  7
This will run forever unless you stop it! Iteration no =  8
This will run forever unless you stop it! Iteration no =  9
This will run forever unless you stop it! Iteration no =  10
```

Explanation: This loop would run forever without the break condition. Always ensure you have a way to stop an infinite loop unless the goal is to run continuously, such as in monitoring systems.

2 Exercise

Exercise 1: Branching with Nested if - Create a program that simulates a simple traffic light system. - Input a color (red, yellow, or green) and output whether cars should "stop", "slow down", or "go". - Extend this by adding another if to check if there are pedestrians present. If yes, even for green light, cars should "wait".

[ ]:

Exercise 2: Multiple for Loops - Write a program that outputs the multiplication tables from 1 to 5. - Use nested for loops to iterate over both the multiplier and the multiplicand.

[ ]:

Exercise 3: Grade Classification - Task: Prompt the user to enter a grade (out of 100). Use if-elif-else statements to classify the grade into A, B, C, or Fail. - Hint: Use branching to evaluate the grade range and print the corresponding result.

[ ]:

Exercise 4: Sum of Even Numbers - Task: Write a program that calculates the sum of all even numbers between 1 and 100 using a for loop. - Hint: Use the range() function with a step value of 2 to iterate through even numbers.

[ ]:

Exercise 5: Bank Withdrawal Simulation - Task: Simulate a simple ATM withdrawal system using a while loop. The user starts with a balance of $1000. The loop should continue to prompt the user for an amount to withdraw, and after each withdrawal, display the remaining balance. Stop the loop when the balance becomes zero or less. - Hint: Make sure the loop exits when the balance is depleted, and handle cases where the withdrawal amount is greater than the available balance.

[ ]: