Lecture 11

Learning outcomes:

- *Python language instructions*
- *Branch organization, cycle organization*
- *The concept of title, value, indicator*

Python language instructions
Branch organization, cycle organization

Python language instructions

- An instruction is a command to the computer, a unit of execution.
- Python code in this case is a set of instructions. It can be presented as a step-by-step recipe.
- Python code is run by an interpreter, a program that executes instructions strictly, line by line.

Branch organization

- Branching in Python refers to the ability to execute certain statements conditionally based on the truth value of certain expressions.

- In other words, it's a way to control the flow of your program based on whether certain conditions are met or not.

- Conditional statements (if, else, and elif) are fundamental programming constructs that allow you to control the flow of your program based on conditions that you specify. They provide a way to make decisions in your program and execute different code based on those decisions.

- Conditional Statements are statements in Python that provide a choice for the control flow based on a condition. It means that the control flow of the Python program will be decided based on the outcome of the condition.

Types of Conditional Statement in Python

1. If Conditional Statement in Python

- If a simple block of code is to be performed if the condition holds then the if statement is used. Here the condition mentioned holds then the block of code runs otherwise not.

- Syntax of If Statement:

```
if condition:
    # Statements to execute if
    # condition is true
```

 # if statement example

```
if 10 > 5:
    print("10 greater than 5")

print("Program ended")
```

✓Output

```
10 greater than 5
Program ended
```

2. If else Conditional Statements in Python

- In a conditional if Statement the additional block of code is merged as an else statement which is performed when if condition is false.

- Syntax of Python If-Else:

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

 # if..else statement example

```
x = 3
if x == 4:
    print("Yes")
else:
    print("No")
```

✓Output

No

3. Nested if..else Conditional Statements in Python

- Nested if..else means an if-else statement inside another if statement. Or in simple words first, there is an outer if statement, and inside it another if – else statement is present and such type of statement is known as nested if statement.

- We can use one if or else if statement inside another if or else if statements.

Nested if..else statement example

```
letter = "A"
if letter == "B":
    print("letter is B")
else:
    if letter == "C":
        print("letter is C")
    else:
        if letter == "A":
            print("letter is A")
        else:
            print("letter isn't A, B and C")
```

✓ Output

```
letter is A
```

4. If-elif-else Conditional Statements in Python

- The if statements are executed from the top down.

- As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed.

- If none of the conditions is true, then the final “else” statement will be executed.

if-elif statement example

```
letter = "A"

if letter == "B":
    print("letter is B")

elif letter == "C":
    print("letter is C")

elif letter == "A":
    print("letter is A")

else:
    print("letter isn't A, B or C")
```

✓ Output

letter is A

5. Ternary Expression Conditional Statements in Python

- The Python ternary Expression determines if a condition is true or false and then returns the appropriate value in accordance with the result. The ternary Expression is useful in cases where we need to assign a value to a variable based on a simple condition, and we want to keep our code more concise — all in just one line of code.

- Syntax of Ternary Expression

Syntax: [on_true] if [expression] else [on_false]

expression: conditional_expression | lambda_expr

 # Python program to demonstrate nested ternary operator

```
a, b = 10, 20

print ("Both a and b are equal" if a == b else "a is greater than b"
        if a > b else "b is greater than a")
```

✓Output

```
b is greater than a
```

Cycle organization

- Cycle (loop) organization is an instruction that involves repeating a block of code multiple times as long as some condition is met.

- The main looping structures in Python are for loops and while loops.

Type of Loops

1. For Loop

- A for loop in Python is used to iterate over a sequence (list, tuple, set, dictionary, and string).

- Example: The preceding code executes as follows: The variable i is a placeholder for every item in your iterable object. The loop iterates as many times as the number of elements and prints the elements serially.

Example of For Loop

```
# iterate through a list
x = ["python", "simplilearn", "tutorial"]
for i in x:
    print(i)
```

✓Output

```
python
simplilearn
tutorial
```

2. While Loop

- The while loop is used to execute a set of statements as long as a condition is true.

- Example: The preceding code executes as follows: We assign the value to variable x as 1. Until the value of x is less than 3, the loop continues and prints the numbers.

Example of While Loop

```
# while loop
x = 1
while x < 3:
    print(x)
    x = x + 1
```

✓Output

1

2

3. Nested Loop

- If a loop exists inside the body of another loop, it is called a nested loop.
- Example: The preceding code executes as follows:

✓The program first encounters the outer loop, performing its first iteration.

✓This first iteration triggers the inner, nested loop, which then runs to completion.

✓Then the program returns to the top of the outer loop, completing the second iteration and again triggers the nested loop.

✓The nested loop runs to completion, and the program returns to the top of the outer loop until the sequence is complete.

Example of Nested Loop

```
# Nested Loop
x = ["simplilearn", "python"]
Num = [1, 2]
for i in x:
    for j in Num :
        print(i, j)
```

✓Output

```
simplilearn 1
simplilearn 2
python 1
python 2
```

The concept of title, value, indicator

The concept of title, value, indicator

- In the context of Python programming, particularly when dealing with data visualization and analysis, the terms "title," "value," and "indicator" can be related to various aspects of data representation.

- These concepts can be widely applied across various domains where data visualization and representation are crucial for analysis, reporting, and decision-making.

- Let's explore each concept and how they can be used in Python:

1. Title

- The "title" typically refers to the title of a plot, graph, chart, or a section of a report. It provides a brief description of what the data or visualization represents.
- Example: Setting a Title in a Matplotlib Plot

```
import matplotlib.pyplot as plt


# Data
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 35]

# Create a plot
plt.plot(x, y , marker='o')

# Set the title
plt.title("Sample Plot of X vs Y")

# Show the plot
plt.show()
```

2. Value

- "Value" generally refers to the data points or numerical values that are being plotted, analyzed, or displayed. Values can come from various sources, such as measurements, calculations, or data sets.
- Example: Displaying Values in a Bar Chart

```
import matplotlib.pyplot as plt

# Data
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]

# Create a bar chart
plt.bar(categories, values)

# Add value labels on top of each bar
for i, value in enumerate(values):
    plt.text(i, value + 0.5, str(value), ha='center')

# Show the plot
plt.show()
```

3. Indicator

- An "indicator" is often used in dashboards or reports to provide a visual representation of a key metric or performance indicator. Indicators can be simple as color-coded symbols, gauges, or any visual cue that highlights a particular data point's status.
- Example: Creating an Indicator with Plotly

```
import plotly.graph_objects as go

# Create an indicator
fig = go.Figure(go.Indicator(
    mode="gauge+number",
    value=270,
    title={'text': "Speed"},
    gauge={'axis': {'range': [0, 500]}}
))

# Show the indicator
fig.show()
```