BACHELOR OF COMPUTER APPLICATIONS

SEMESTER *5*

DCA3104

PYTHON PROGRAMMING

Unit 5

Branching and Looping in Python

Table of Contents

1. INTRODUCTION

In the previous units, we learned about what a programming language is and how it is used to communicate with a computer. Programming languages are used to create programs that control the actions of a computer and to implement algorithms efficiently.

You were also introduced to the Python language, which is a general-purpose programming language. Python is used to create simple algorithms, perform mathematical and engineering calculations, and handle large amounts of data. Unlike other programming languages, Python has a relatively simple syntax that allows a programmer to create programs with a few lines of code.

Python's simple and easy-to-understand structure has made it popular among beginner and professionals. Its versatile nature has made Python the most used programming language for the internet's most popular websites.

In this unit, you will be introduced to different types of conditional statements, types of loops and all the conditions, variables, and statements used in them.

Conditional statements are lines of code that help execute different instructions based on certain conditions. These statements assist a programmer to direct the flow of code execution and control how an application interacts with a user. Conditional statements are also known as Branching. When a set condition is checked and met, the program may branch off in one way or another and proceed in the relevant direction.

Loops, on the other hand, are sets of code that are repeated as long as a condition is valid or a sequence is completed.

As you may recall, keywords are reserved words used in Python that cannot be assigned any value. In this unit, you will use the if, else, pass, etc., keywords. You will also use different operators like arithmetic operators, assignment operators, logical operators, comparison operators, etc., in loops and branching.

1.1 Learning Objectives

Upon completing this chapter, you will be able to:

- *Use the Python IF, If-Else, ELIF, and ELSE statements to write and read structured programs.*
- *Implement simple loop structures using the FOR and WHILE Loops to read and write programs.*
- *Use nested and chained conditional statements.*
- *Construct algorithms that use conditional and loop structures.*

2. PYTHON IF-ELSE CONDITIONS

Conditional expressions in Python are logical expressions where logical operators compare, evaluate, or check if the input (operand) meets the conditions and returns 'True'. The block of code gets executed based on this. If the expression returns a 'False', the block does not execute, and the cursor moves on to the next line of code.

Python follows the common logical conditions/operators used in mathematics. These include:

- **Equals to (==) -** x==y - 'True' if value of x equals value of y. Otherwise, it is 'False'.
- **Not equal to (!=)** - x!=y - 'True' if the value of x does not equal the value of y. Otherwise, it is 'False'.
- **Less than (<)** - x<y - 'True' if the value of x is less than the value of y. Otherwise, it is 'False'.
- **Less than or equal to (<=) -** x<=y - 'True' if the value of x is less than or equal to the value of y. Otherwise, it is 'False'.
- **Greater than (>) -** x>y - 'True' if the value of x is more than the value of y. Otherwise, it is 'False'.
- **Greater than or equal to (>=) -** x>=y - 'True' if the value of x is more than or equal to the value of y. Otherwise, it is 'False'.

2.1. Indentation In Python

In the above structures of *if* and *if...*Else statements, you might have observed '...' after the conditional expression. It represents an indent or indentation in code. A unique feature of Python is the indentation of the block code away from the conditional expressions that are equal to four spaces. All the statements in a block must have the same indentation. When you wish to exit the conditional statement, hit the **Enter** key to move to the succeeding line and then the **Backspace** to realign with the conditional statement line.

STUDY NOTE

During programming, an indentation can be achieved using the Tab key on the keyboard.

Most programming languages require that their conditional blocks are enclosed in parentheses, curly brackets etc. In Python, this is not required. However, a semi-colon (':') must follow the conditional statement in the same line, and the following statements should have a proper indentation.

SELF-ASSESSMENT QUESTIONS - 1

1. Conditional statements are also known as _____.
2. According to the Python guidelines, the indentation of the body is equal to _____.
3. In Python, indentation is represented as _____.
4. Python provides a unique set of operators for coding. [True/False]
5. No programming language requires their block of code to be enclosed in brackets. [True/False]
6. '!=' returns a 'False' value if one variable is equal to the other. [True/False]
7. All conditional statements should end with a:

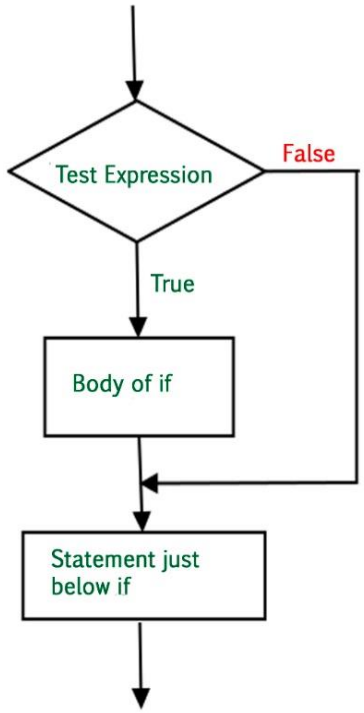
   a) colon (:) b) semi-colon (;)

   c) comma (,) c) period (.)

3. THE IF-ELSE STATEMENT

'If' is the simplest conditional statement in Python. When the specified condition of the 'if' statement is met: the block of code or compound statement under it is executed. The 'if' statement has the following structure in Python:

```
if < conditional expression > :
... < block >
```



Source: geekforgeeks.org

Fig 1: Decision Path of if...else Statement

Example:

```
T = float(input('What is the water temperature? '))
if T > 24:
        print('Water is safe!')
        print('Jump in!')
# First line after if part
```

In the above example, Python will ask the user to input the water temperature. If the temperature is more than 24, the condition is 'True', and the print commands are executed. If the temperature is less than 24, the condition is 'False'. The program moves to the code after the if block without executing anything. In other words, the program gives no response when the temperature is below 24.

STUDY NOTE

'#' represents a comment in the code. The comment statements are not executed but are only written to understand the code properly.

[Note that we need to convert the string input to float before we assign it to 'T']

However, as a programmer, you will want your application to perform a set of steps when a condition is met (True statement) and another set of steps when it is not (False statement). We use the *if...Else* statement to execute different blocks of code based on the True or False criteria.

The if...Else structure is as follows:

```
if < conditional expression > :
... < block >
else :
... < block >
```

Example:

We can use the same example of temperature mentioned above:

```
T = float(input('What is the water temperature? '))
if T > 24:
        print('Water is safe!')
        print('Jump in!')
else:
        print('Better not go in!')
# First line after if-else
```

When the if condition evaluates to 'False', the print command in the else part is executed automatically. It is only logical because if the first condition is false, then the second one is true.

3.1. The Elif Statement

STUDY NOTE

There is no limit to the number of elif statements that can be used in one block of code.

The branching execution of the *elif* (short for else if) statement is based on several alternatives. It is also known as the if...*elif*...else statement and involves using many *elif* statements. Python evaluates all the *elif* statements in succession and only executes the block under the first elif expression. In other words, it does not matter if there are more than one 'True' conditions. Python stops at the first 'True' value, executes it, and then exits the if...*elif*...else structure.

In case no *elif* statement returns the "True" value, the else block is executed. In most cases, however, the *else* expression is not required and can be omitted. It is up to the programme to determine whether or not an *else* expression is required in the code.

You can use the elif expression in two ways:

A. Chained Conditionals

Using elif in the chained conditional expressions is quite straight forward as mentioned above and has the following syntax:

```
if < conditional expression > :
... < block >
elif < conditional expression > :
... < block >
elif < conditional expression > :
 ... < block >
elif < conditional expression >:
 ... < block >
elif < conditional expression > :
 ... < block >
:
:
```

```
:
 else :
... < block >
Example:
if choice == 'a':
... print("You chose 'a'.")
elif choice == 'b':
... print("You chose 'b'.")
elif choice == 'c':
... print("You chose 'c'.")
else:
... print("Invalid choice.")
```

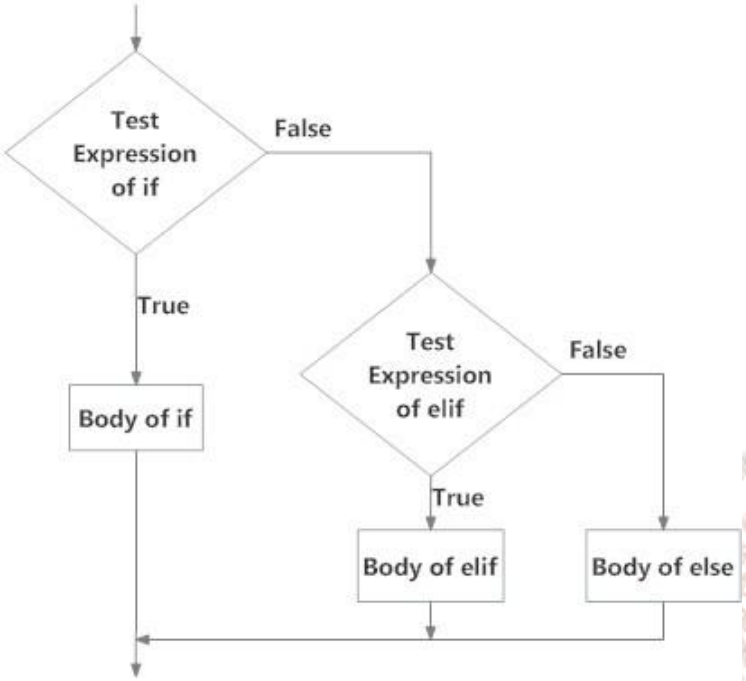
Each condition is checked in the given order. If the second elif statement returns 'True', "You chose 'b'" is printed and the cursor exits the chained structure to move to the next code line.

B. Nested Conditionals

In Nested conditionals, we can ***nest*** a set of conditional expression within another. It has the following syntax:

```
if < conditional expression > :
... < block >
else:
... if < conditional expression > :
   ... < block >
... else :
    ... < block >
```

The nested format has an outer conditional with two branches. The second branch has an if expression with two branches of its own, and so on. Although this format has good structure with indentation, it becomes very difficult to read nested expressions in the long run. Hence, it is better to avoid this when you can.



Source: thecrazyprogrammer.com

Fig 2: Decision Path of if…elif…else statement

The pass Statement

You may encounter situations where nothing has to be done if a condition returns 'True'. Since Python does not permit an empty block in the code, we use the "pass" statement. It is written as:

```
if < conditional expression > :
... pass # don't do anything
```

Composite Expressions

To combine two conditional expressions, we use the logical operators: "and", "or", or both to make a **composite conditional expression**.

Example:

```
if < conditional expression > and < conditional expression > :
... < block >
OR
```

if < conditional expression > **or** < conditional expression > :

... < block >

Activity I

Write a Python program that prompts the user for a string and calculates the number of letters and digits separately. Use the if...elif...else statement for this program.

SELF-ASSESSMENT QUESTIONS - 2

8. A block of code should always have an *else* statement. [True/False]
9. The pass statement is executed when the given condition is satisfied. [True/False]
10. Any number of *elif* statements can be used in a code. [True/False]
11. Two expressions are concatenated to form a _____.
12. *elif* stands for _____.
13. if x >= 0 :

        print ('Your number is a whole number')

    The above code will always give a:

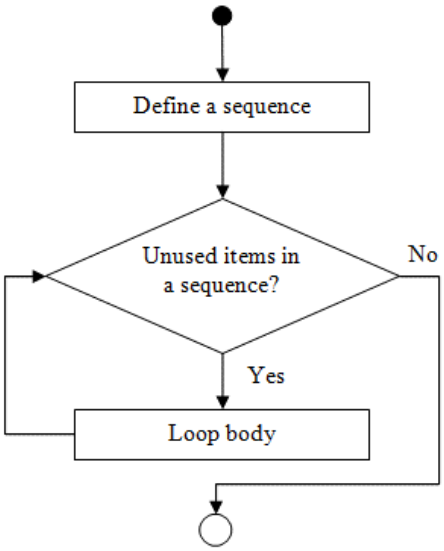
    a) positive integer b) negative integer

    c) any number d) cannot say

4. PYTHON LOOPS

The process of looping or iteration is fundamental to all programming languages and involves the repetition of tasks. A loop is a set of instructions (block) repeatedly executed as long as the defined conditions are met or satisfied. In Python, we use two statements for looping: the "for" statement and the "while" statement.

4.1. For Loop



Source: teststeststests.com

Fig 3: Decision Path of For Loop

The *for* loop executes each statement in a sequence and is hence used with Python's **iterables** that include: a string, list, tuple, dictionary and set. Unlike in the *while* loop, you do not have to set a looping variable in the *for loop*. The structure of *for loop* in Python is:

```
for < var > in < iterable > :
... < statement >
... < statement >
```

In the above format, <var> represents the looping variable that updates to the next value of the iterable < iterable > each time the loop is executed.

Unlike the *for* loop in other programming languages, the Python *for* loop can incorporate the *break statement*, *continue* statement and the *else* clause.

STUDY NOTE

In Python, the *break* statement, *continue* statement and the *else* clause can be nested in looping or branching without any restrictions.

As looping is controlled by the number of items in the iterable, infinite loops cannot likely be formed. However, the variable can accidentally be manipulated in the body of the loop, resulting in an infinite loop. Hence, programmers should check, vet and recheck the code before execution.

4.1.1. Using Sequence

As mentioned above, loops use a set of items called iterables to execute statements. A **Sequence** is a generic term used to refer to iterables. Python has many types of sequences; however, the most common types are:

- **Lists -** These are the most versatile sequences. The elements of a list can be any object that can be added, reassigned or removed.
- **Tuples -** These sequences are similar to lists. However, they cannot be changed.
- **Strings -** These are special sequences that can only store characters and have a unique notation. Apart from characters, sequence operators can also be stored as a string.

Sequence Operators

- '+' operation combines two sequences through concatenation.
- '*' operation repeats a sequence a given number of times. Example: x*n; sequence x will add with itself 'n' number of times.
- x will return 'True' in Seq if it is an element of Seq. Otherwise, it will return 'False'.
- seq[i] will return the ith item in the sequence. You should note that Python starts with the '0' index, that is, the first has index 0, the second has index 1, and so on.
- seq[-i] will return the ith element from the end of the sequence.

4.1.2. Using Range

The *range* function is used to repeat or loop a block of code a given number of times. We know that Python is a **zero-indexed** programming language. Hence, by default, it returns a sequence of elements starting from zero index. The index sees an **increment** by 1 till the set range is reached.

The *range()* function is used as:

```
for < variable > in range( < number > ) :
... < block >
```

The loop will run from a 0 value variable to a value equal to the number specified minus one.

Example:

```
for x in range( 5 ) :
... < block >
```

The loop will run from x=0 through to x=4 with increments of 1.

The starting value of a range() be change from the default 0:

```
for x in range( 2, 5 ) :
... < block >
```

The loop will run from x=2 through to x=4 with increments of 1.

The increment values can also be changed as follows:

```
for x in range( 5, 16, 2 ) :
... < block >
```

The loop will run from x=5 through to x=15 with increments of 2.

Activity II

Write a Python program that asks the users for a number and give the number's multiplication table a output. The table should be from 1 to 20. You can use the range function in the for loop.

SELF-ASSESSMENT QUESTIONS - 3

14. _____ is a set of statements that are repeatedly executed.
15. The looping expressions used in Python are _____ and _____.
16. Sequences are also referred to as _____.
17. String sequences store numbers. [True/False]
18. It is impossible to form infinite loops in Python. [True/False]
19. The Operation seq[i] will return the ith item in a sequence. [True/False]
20. In Python, the index numbers start from ___ by default.

    a) 1 b) 0 c) any number d) cannot say

21. The following code will give you values of x from:

```
for x in range( 1, 9 ) :
... < block >
```

    a) 1 to 9 b) 1 to 7

    c) 1 to 8 d) 0 to 8

5. NESTED FOR LOOPS IN PYTHON

STUDY NOTE

Most programmers limit loop nesting to only two to three loops to avoid confusion.

Similar to the nested if conditional statements, programmers can also nest for loops within each other for better functionality. Python gives programmers the flexibility to nest loops in conditional statements, composite conditional statements and vice versa without any restrictions. The nesting expression can be written as follows:

```
for [first iterating variable] in [outer loop]: # Outer loop
        ... < block > # Optional
for [second iterating variable] in [nested loop]: # Nested loop
        ... < block >
```

The program evaluates the outer loop first, executing the first iteration. The iteration triggers the next (nested) loop to completion. The program returns to the top of the outer loop, completes the second iteration, and triggers the nested loop again. The loop continues till the sequence is complete or there is a disruption in the process.

Let's take a look at an example to understand how we can nest for loops in Python:

```
num_list = [1, 2, 3]
alpha_list = ['a', 'b', 'c']
for number in num_list:
... print(number)
... for letter in alpha_list:
   ...print(letter)
```

For this program, we get the output as:

```
#Output
1
a
b
c
2
```

```
a
b
c
3
a
b
c
```

From the above output, we can see that the program executed the first iteration to print '1', goes to the nested loop to print 'a, b, c' consecutively. Once the inner loop is complete, it goes back to print '2' in the outer loop, and again prints 'a, b, c' by triggering the inner loop, so on.

5.1. Using Else Statement In Loop

As mentioned earlier, Python gives a programmer the freedom to nest and also use various conditional statements in loops. The *else* statement is one such statement that can be used in loops even without the *if* conditional statement.

However, *else* is only functional when the loop is terminated normally. If the loop is terminated forcefully, the else statement is overlooked and not executed.

STUDY NOTE

'\' is called the line-break character. It is used to break a long line of code into multiple lines. Hence, it helps programmers understand the code more easily.

The else statement can be executed in the *for loop* as follows:

```
for < var > in < iterable > :
... < statement >
else :
... < statement >
```

Let's look at an example:

```
for currentLetter in 'Hi everyone!':
... print ('The current letter is', currentLetter)
else:
... print ('All letters were printed.')
```

```
#The output will be:
>>>
The current letter is H
The current letter is i
The current letter is
The current letter is e
The current letter is v
The current letter is e
The current letter is r
The current letter is y
The current letter is o
The current letter is n
The current letter is e
The current letter is !
All letters were printed.
>>>
```

We got the above output because the loop iteration ended normally and the else statement was executed.

Let's see what happens when the loop does not end normally:

```
for currentLetter in 'Hi everyone!':
  print ('The current letter is', currentLetter)
  if currentLetter == 'r':
     break
else:
  print ('All letters were printed.')

#Output:
>>>
The current letter is H
The current letter is i
The current letter is
```

```
The current letter is e
The current letter is v
The current letter is e
The current letter is r
>>>
```

The *for* loop terminated as soon as the letter 'r' was encountered due to the *if* condition provided. The break statement disrupts the code abruptly. Hence, the *else* statement did not execute as the loop did not complete normally.

Activity III

Write a Python program that finds the numbers between 1000 and 2700, that are divisible by 7 and are also multiples of 5. Use a nested for loop.

SELF-ASSESSMENT QUESTIONS - 4

22. A nested loop is triggered only after the outer loop is executed. [True/False]
23. The else expression is executed even if there is a disruption in the code. [True/False]
24. The else statement can be used in a loop even without the _____ statement.
25. _____ character is used to break a long line of code.
26. The output of the following code is:

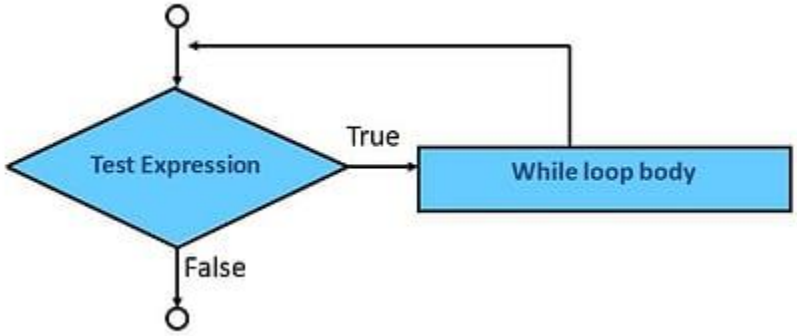
```
for i in range(10):
        if i == 5:
            break
        else:
            print(i)
    else:
print("Here")
```

a) 0 1 2 3 4 b) 1 2 3 4 c) 0 1 2 3 d) 1 2 3

6. WHILE LOOP

The while loop is used to repeatedly execute a block of code as long as the given condition is satisfied. The while loop has the following syntax:

```
while < expression > :
... < block >
```



Source: codestown.com

Fig 4: Decision Path of While Loop

The expression controls the loop execution and involves looping variable(s) with logical operations. The looping expression sets the condition based on which the loop will continue to run or terminate.

While executing an assignment statement in the *while* loop, the right-hand side of the expression is evaluated first. The evaluated result is then assigned to the variable. In loop expressions, the variable is evaluated and repeatedly updated while the given condition is 'True', otherwise it is terminated.

STUDY NOTE

In Python, any non-zero value is viewed as 'True' while 'None' and 0 are interpreted as 'False'.

An Example of a *while* loop:

```
x = 5
while x < 10:
... print (x)
... x+=1
#Output:
>>>
5
6
7
8
9
>>>
```

The statement ***x+=1*** adds **1** to **x** in each loop iteration. This ensures that the loop does not become an endless loop.

Activity IV

Write a program that finds the Fibonacci series between 0 to 50. Every next number in the Fibonacci sequence is found by add the previous two numbers

SELF-ASSESSMENT QUESTIONS - 5

27. The output of the following code is _____.

```
i = 1
while True:
        if i%3 == 0:
                break
print(i)
i + = 1
```

28. The output of the following code is:

```
i = 0
while i < 5:
        print(i)
        i += 1
 if i == 3:
                break
else:
        print(0)
```

a) 0 1 2 0 b) 0 1 2 c) 1 2 3 d) 0 1 2 3 4

29. In loops, the value on the right-hand side is evaluated first and if it meets the condition, it is assigned to the variable. [True/False].

30. _____ are used with operators to make loop expressions.

31. The while loop executes the loop expression when the variable evaluates to _____.

7. BREAK AND CONTINUE

7.1. The Break Statement

In Python, the break statement is used to abruptly terminate the loop execution. It is mostly used under the if statement that sets the condition to break out of the loop. Once the program breaks out of the loop, it resumes the execution at the first line after the loop.

STUDY NOTE

If a break statement is written inside a nested loop (loop within a loop) it terminates only the inner loop.

The break statement is used as:

```
while < expression > :
        < statement >
        < statement >
        :
        break
        :
        < statement >
< block >
```

Example:

```
word = input ('Type a word that is less than 5 characters long: ')
letterNumber = 1
for i in word:
  print ('Letter', letterNumber, 'is', i)
  letterNumber+=1
  if letterNumber > 5:
      print('The word you have entered is more than 5 characters long!')
      break
```

The above code will print each letter of the word you have entered. However, if the character count exceeds 5, the break statement in the code will terminate the code after 5 characters.

Type a word that is less than 5 characters long: Avalanche

Letter 1 is A

Letter 2 is v

Letter 3 is a

Letter 4 is l

Letter 5 is a

The word you have entered is more than 5 characters long!

7.2. The Continue Statement

Similar to the break statement, the continue suddenly stops the running iteration. However, instead of terminating the loop, it goes back to the loop expression for re-evaluation before continuing. If the re-evaluated results do not comply with the expression, the loop terminates and the program moves to the next line after the body of the loop. If the expression is still valid, the program continues with the next iteration.

STUDY NOTE

Both the break and continue statements are used to terminate the current iteration or loop without checking the test expression

The continue statement is used as:

```
while < expression > :
        < statement >
        < statement >
        :
        continue
        :
        < statement >
< block >
```

Example:

```
for i in range(6,15):
   if i==9:
      continue
   print (i)
```

```
#Output:
>>>
6
7
8
10
11
12
13
14
>>>
```

The program prints all numbers between 6 to 15 except 9. When the variable i is equal to 9, the if statement is executed and the continue statement under it will force the program to skip the print statement before it moves to the next iterable.

Activity V

Create a Python code that prints all numbers between 0 to 30, except 10 and 15. Use the continue statement in the code.

SELF-ASSESSMENT QUESTIONS - 6

32. To abruptly terminate a code, we use the _____ statement.
33. When we use the _____ statement, the cursor/program goes back to the conditional expression for re-evaluation.
34. If a break statement is written inside a nested loop (loop within a loop) it terminates only the inner loop. [True/False]
35. The break and continue statements can only be used in the while loop. [True/False]
36. The output for the following code is:

```
    True = False
while True:
        print(True)
        Break
```

A "True" b) "False" c) none of the above d) "None"

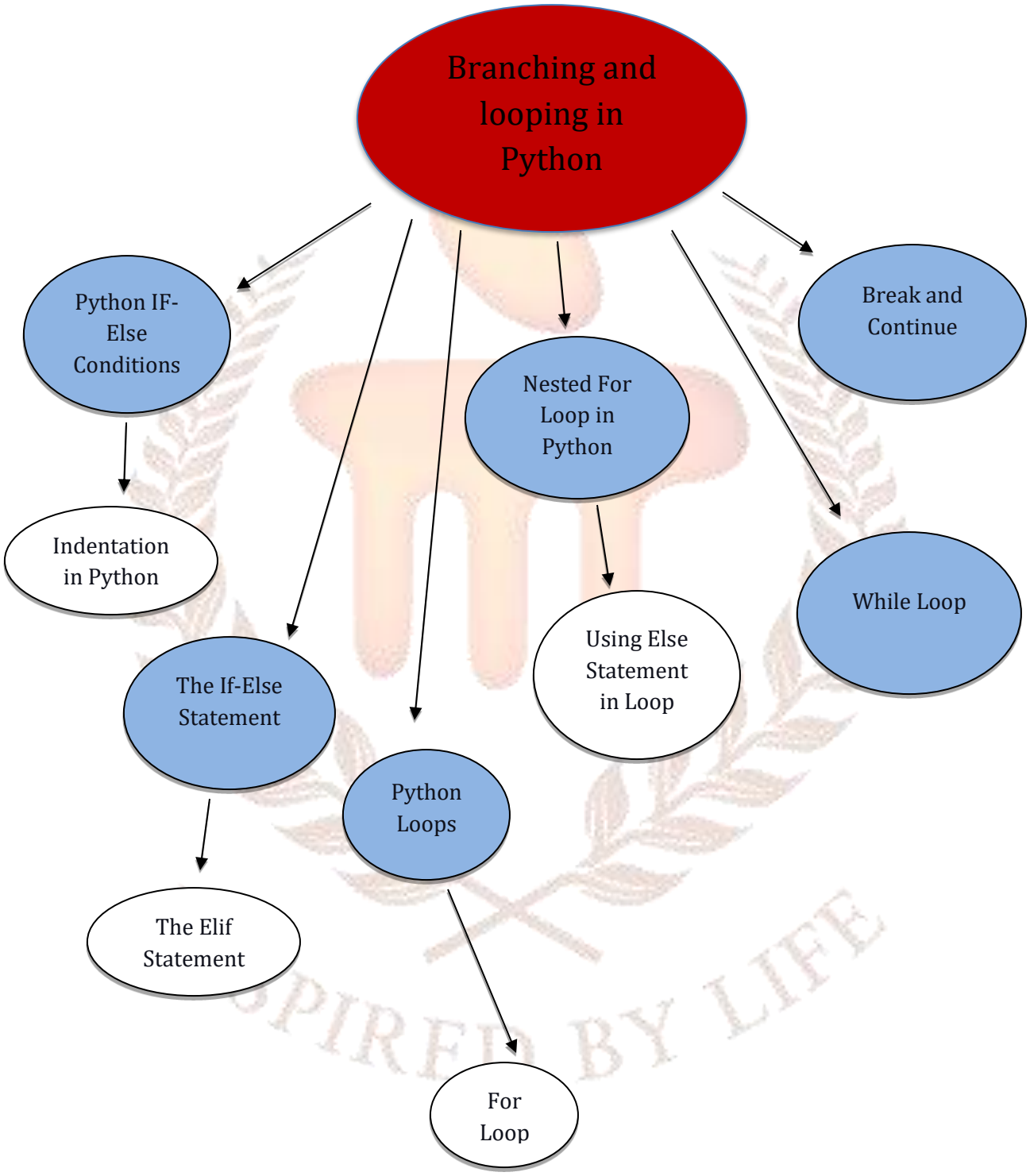
8. SUMMARY

- Branching or conditional statements are used to set the parameter for the execution of a code.
- Conditional expressions are logical and follow operations similar to those in mathematics such as equal to (==), less than (<), greater than (>), less than or equal to (<=), greater then or equal to (>=), not equal (!=), etc.
- The if...else expression is used to execute an action in the case of both 'True' and 'False'. One block of code is executed if the condition is ‘True’ and another block is executed if it is ‘False’.
- The conditional statements may be executed in a sequential manner (chained conditionals) or different conditional may be nested within a conditional statement (nested conditionals).
- The pass statement is used instead of an empty block in Python. If the condition is ‘True’ and a pass statement is given, no action is executed.
- The composite conditional expression can be constructed using "and" and "or". The ‘and’ operator returns ‘True’ if both conditions are met, while the ‘or’ operator returns ‘True’ if even one condition is satisfied.
- Python gives programmers the freedom to nest loops and statements in any combinations without any restrictions.
- Looping keeps executing a code block until the conditions are evaluated as 'False'.
- The looping expression sets the condition for which the loop will continue to run. Unlike the while loop, the for loop does not require a looping variable for execution.
- Sequences are collections of items or elements that can be iterated over using a loop. Lists, tuples, strings, etc., are called sequences. We can perform tasks on sequences using operators or functions like ‘+’, ‘*’, seq[i], etc.
- The range() function sets a limit or range using which a loop returns values. Range() uses indexing to retrieve items.
- Python indexing starts from ‘0’.
- In nested loops, the else statement can be used even without the if statement.

- The break statement is used to abruptly exit the code, while the continue statement is used to go back and re-evaluate the conditional expression.

9. GLOSSARY

- **Block -** A set of statements that have the same indentation.
- **Branch -** One of the possible paths for execution that is determined by the conditional expressions.
- **Compound statement -** Python code is made up of a header and a body. The header starts with a keyword and ends with a colon (:). The body is the series of indented statements under the header.
- **Infinite Loop -** The loop in which the termination criteria is never met.
- **Loop variable -** A variable used as a part of the termination condition in a while loop.
- **Nesting -** When one program structure is included within another.
- **Newline** - A special character that makes the cursor move to the start of the next line.
- **Prompt -** A suggestion that asks users to input data.
- **Reassignment -** Assigning more than one value to a variable during program execution.
- **Iterables -** A set of programming objects used in iteration (repetition of a process to produce an outcome).
- **Increment -** Increasing the value of an object by 1.
- **Decrement -** Decreasing the value of an object by 1.


Branching and looping in Python
Python IF-Else Conditions
Indentation in Python
The If-Else Statement
The Elif Statement
Nested For Loop in Python
Using Else Statement in Loop
Python Loops
For Loop
Break and Continue
While Loop

Automating Data Tasks With Python Loops

Source: https://bit.ly/3dbKa1Z

A new accounting firm saw a rise in their clientele only a few months after their establishment. They realized that the traditional method of maintaining client accounts through MS Excel would no longer be feasible. In order to not loose their clientele, the firm decided to use Python programming language to automate data tasks.

They used Python loops to iterate over the data in lists and data frames. This helped them to automatically calculate and convert the values in the lists and database. The system set up by the firm successfully retrieved client data, filtered it and organized it for analysis. The conditions for this were set using the for and while loops.

Python provides a simple and clean coding syntax and the accounting system proved to be quite affective for performing data tasks in a less amount of time.

Discussion Questions

1) Discuss the need for automation in sectors like accounting.
2) To what extend is automated accounting effective?
3) Discuss the possible conditions the firm set in the loops.

10. TERMINAL QUESTIONS

SHORT ANSWER QUESTIONS

1)Write the output of the given program:

```
for val in "stranger":
    if val == "a":
    continue
    print(val)
print("The end")
```

2) What do you understand by control flow statements in Python?
3) Write about the loop interruption statements in Python.

LONG ANSWER QUESTIONS

4) Write a code that accepts string input and calculates the number of character and digits.
5) Write a nested loop statement that will prompt the user to input a number and print the range of numbers from 1 to the input number.
6) Write a program that will prompt the user to input a number and give a sum of all the natural numbers up to the input number.
7) Using the if...else condition, write a program that will let people below 18 and above 60 know they cannot ride a rollercoaster.
8) The following code will give the output:

```
for val in "star":
    if val == "a":
    break
    print(val)
print("The end")
```

9) Write a program to calculate the sum of the following numbers:
5, 3, 7, 2, 6, 9, 4, 8, 1.

11. ANSWERS

A. SELF ASSESSMENT QUESTIONS

1) Branching
2) Four spaces
3) '...'
4) False
5) False
6) True
7) a) colon (:).
8) False
9) True
10) True
11) composite conditional expression
12) else if
13) a) Positive integer
14) loop
15) for, while
16) iterators
17) False
18) False
19) True
20) b) 0. Python is a zero-index language
21) c) 1 to 8.
22) True
23) False
24) if
25) '\n'
26) a) 0 1 2 3 4
27) error! Explanation: There should be no space between = and +
28) b) 0 1 2. else statement is not executed as loop breaks.

29)True

30)looping variables

31)“True”

32)break

33)continue

34)True

35)False

36)c) none of the above. “True” and “False” are reserved words in Python.

B. TERMINAL QUESTIONS

1) Answer Output:

   s
   t
   r
   n
   g
   e
   r
   The end

   All the characters up till 'a' are printed. If the string is 'a', the block is not executed. This is because the continue statement is executed when 'val' is 'a'. The program then moves to the next iterable, that is, 'n' and continues the loop. Hence, we see all characters, except 'a' are printed.

2) The statements used to manipulate or change the execution flow of a program are known as control flow statements. A typical Python code sequentially executes a program (from top to bottom) but some statements in Python can break this order of execution. They include loops, branching, etc.
3) Loop interruption statements let users terminate a loop iteration without letting the loop run its full course. There are two loop interruption statements:

break statement: This statement terminates the loop completely and shifts the control flow directly outside the loop.

continue statement: This statement terminates only the current loop iteration and shifts the control flow to the next iteration of the loop.

4) Code answer:

```
s = input("Input a string")
d=l=0
for c in s:
  if c.isdigit():
    d=d+1
  elif c.isalpha():
    l=l+1
  else:
    pass
print("Letters", l)
print("Digits", d)
```

5) Code Answer:

```
x = int(input('Enter a number: '))
while x != 0:
  for y in range (1, x):
    print (y)
    y+=1
  x = int(input('Enter a number: '))
```

6) Code Answer:

```
# Program to add natural
# numbers up to
# sum = 1+2+3+...+n


# To take input from the user,
n = int(input("Enter n: "))
```

```
# initialize sum and counter
sum = 0
i = 1

while i <= n:
   sum = sum + i
   i = i+1 # update counter
# print the sum
print("The sum is", sum)
```

7) Code Answer:

```
x = int(input('Enter your age: '))
if x < 19 or x > 60:
   print('You are too young or too old to ride!')
else:
   print('Welcome, you are of the right age!')
```

If the user enters an age between 19 and 60, the program will print "Welcome, you are of the right age". On the other hand, if the age is below 19 or above 60, the program will tell the user they are too young or too old.

8) Output:

s
t
The end

In this program, we iterate through the "star" sequence. We check if the letter is 'a', then we break from the loop. Hence, we see in our output that the letters till 'a' (s and t) get printed. After that, the loop terminates.

9) Code Answer:

```
# Program to find the sum of all numbers stored in a list
# List of numbers
numbers = [5, 3, 7, 2, 6, 9, 4, 8, 1]
```

```
# variable to store the sum
sum = 0
# iterate over the list
for val in numbers:
   sum = sum+val
print("The sum is", sum)
```

12. SUGGESTED BOOKS AND E-REFERENCES

BOOKS

- Lutz. M, Python pocket reference: Python in your pocket.
- John. M. Z. Python Programming: An Introduction to Computer Programming

E-REFERENCES

- Python, viewed on 25 March 2021,

  <https://geek-university.com/python/the-if-statement/>

- Beginning Python Programming for Aspiring Web Developers, viewed on 25 March 2021,< http://www.openbookproject.net/books/bpp4awd/>
- Python 3, viewed on 25 March 2021

  <https://www.programiz.com/python-programming/tutorial >