Name: Chandana Ramesh Galgali
Batch: P6-1 Roll No.: 16010422234
Experiment / ~~assignment~~ / ~~tutorial~~ No. 03
Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

TITLE: Decision Making Statements

AIM: 1) Write a program to count the number of prime numbers and composite numbers entered by the user.

2) Write a program to check whether a given number is Armstrong or not.

Expected OUTCOME of Experiment: Use different Decision Making statements in Python.

Resource Needed: Python IDE

Theory:

Decision Control Statements

1) Selection/Conditional branching statements

a) if statement
b) if-else statement
c) if-elif-else statement

2)Basic loop Structures/Iterative statement

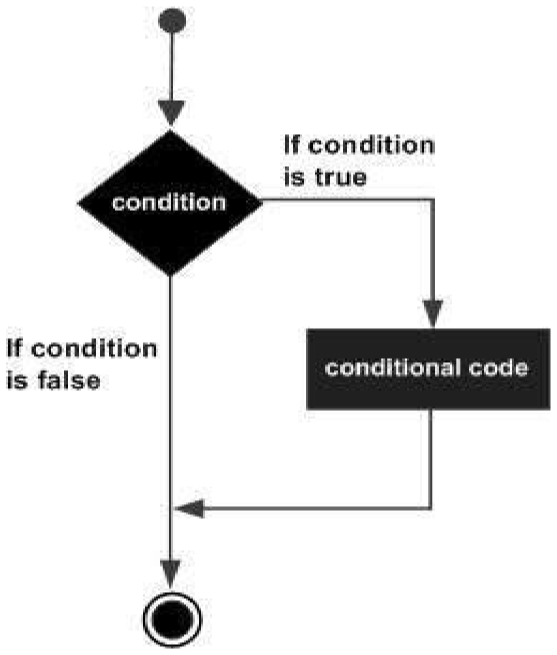
a) while loop
b) for loop

If statement:

In Python **if** statement is used for decision-making operations. It contains a body of code which runs only when the condition given in the **if** statement is true.

Syntax:
```
if condition:
   statement(s)
```

If flowchart:

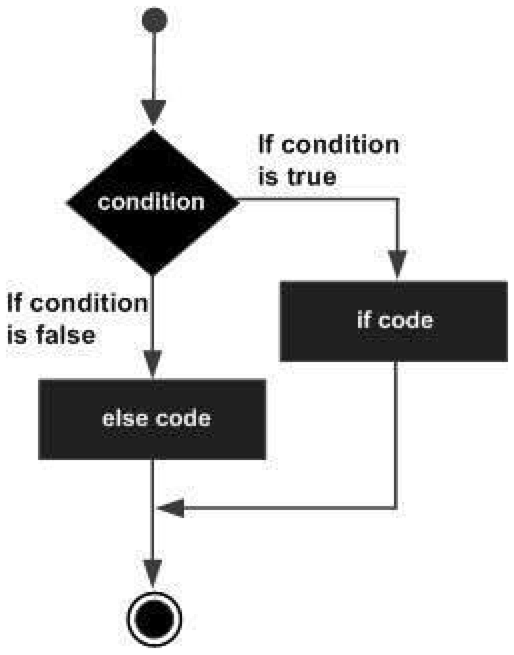


If-else Statement:

An **else** statement can be combined with an **if** statement. An **else** statement contains the block of code that executes if the conditional expression in the **if** statement resolves to 0 or a FALSE value.
The **else** statement is an optional statement and there could be at most only one **else** statement following **if**.

Syntax:
```
if expression:
   statement(s)
else:
   statement(s)
```

If-else flowchart:



If-elif-else Statement:

The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.
Similar to the else, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if.**

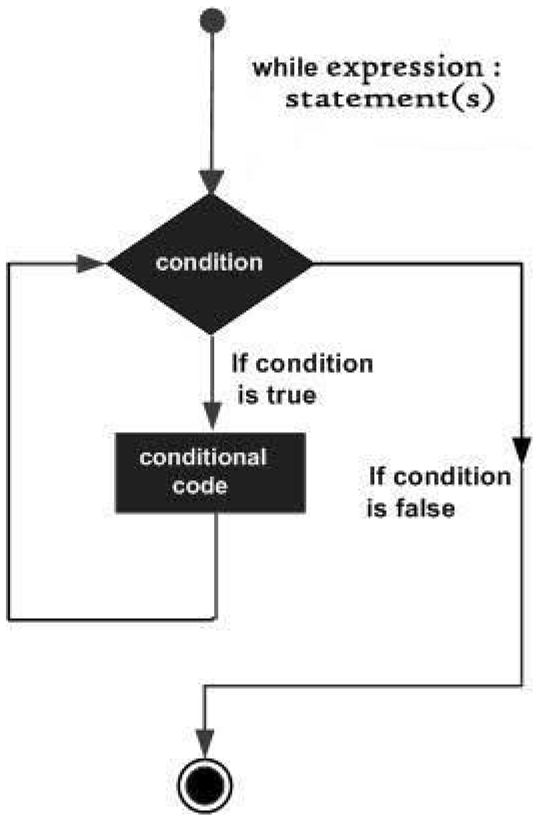
Syntax:
```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

While loop:

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:
```
while expression:
   statement(s)
```

While loop flowchart:

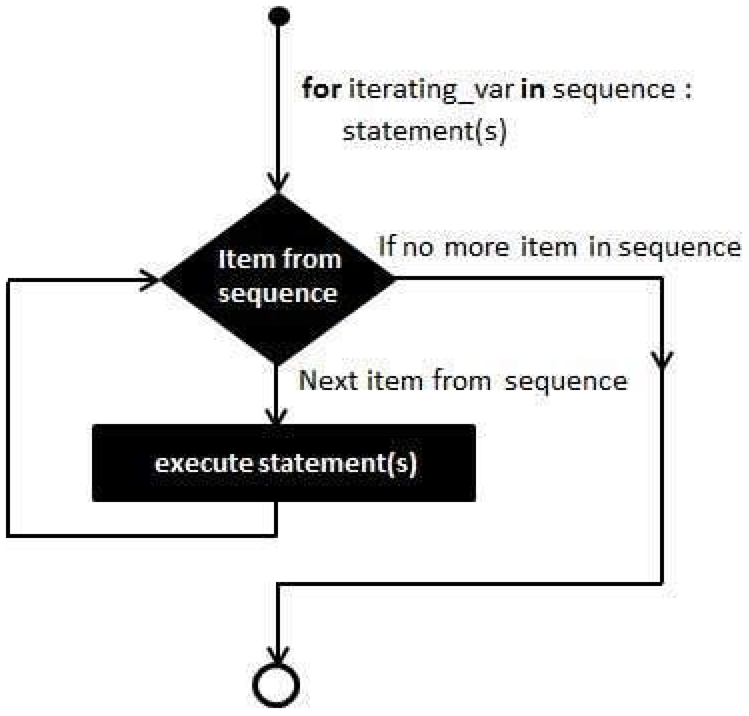


For Loop:

The **for** statement in Python differs a bit from what you may be used to in C. Rather than giving the user the ability to define both the iteration step and halting condition (as C), Python’s **for** statement iterates over the items of any sequence (a list or a string), in the order that they appear in the sequence.

Syntax:
```
for iterating_var in sequence:
   statements(s)
```

For loop flowchart:



Problem Definition:

1) Write a program to read the numbers until -1 is encountered. Also, count the number of prime numbers and composite numbers entered by the user

2) Write a program to check whether the number is an Armstrong or not. (Armstrong number is a number that is equal to the sum of cubes of its digits for example: 153 = 1^3 + 5^3 + 3^3.)

Books/ Journals/ Websites referred:

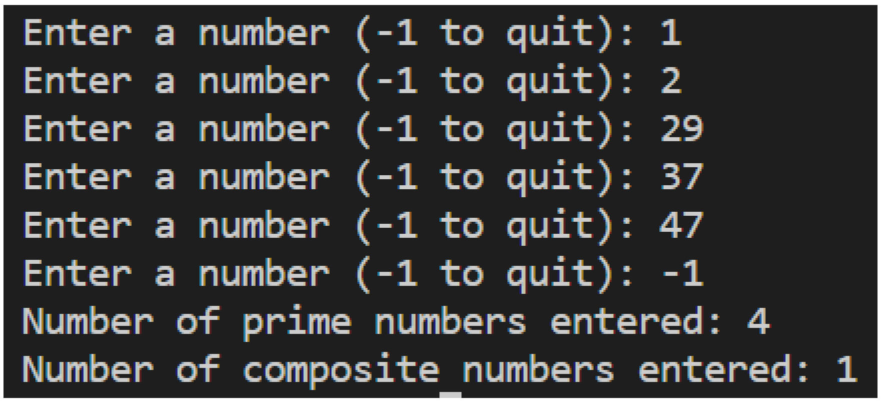
1. Reema Thareja, *Python Programming: Using Problem Solving Approach*, Oxford University Press, First Edition 2017, India
2. Sheetal Taneja and Naveen Kumar, *Python Programming: A modular Approach*, Pearson India, Second Edition 2018,India
3. https://docs.python.org/3/tutorial/controlflow.html#for-statements

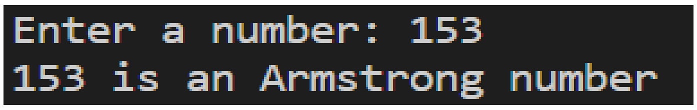
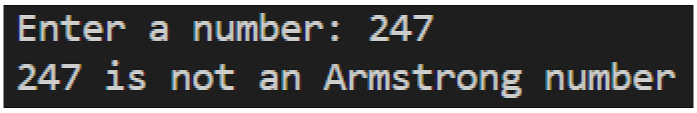
Implementation details:

1.
```
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True
# Initialize counters
prime_count = 0
composite_count = 0
# Read numbers from user until -1 is entered
while True:
    num = int(input("Enter a number (-1 to quit): "))
    if num == -1:
        break
    elif is_prime(num):
        prime_count += 1
    else:
        composite_count += 1
# Print the results
print(f"Number of prime numbers entered: {prime_count}")
print(f"Number of composite numbers entered: {composite_count}")
```

2.
```
num = int(input("Enter a number: "))
num_of_digits = len(str(num))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** num_of_digits
    temp //= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

Output(s):

1.

```
Enter a number (-1 to quit): 1
Enter a number (-1 to quit): 2
Enter a number (-1 to quit): 29
Enter a number (-1 to quit): 37
Enter a number (-1 to quit): 47
Enter a number (-1 to quit): -1
Number of prime numbers entered: 4
Number of composite numbers entered: 1
```

2.

```
Enter a number: 153
153 is an Armstrong number
```

```
Enter a number: 247
247 is not an Armstrong number
```

Conclusion:
On successful completion of the experiment, we learnt about the use of decision making statements in Python.

Post Lab Questions:

1) When should we use nested if statements? Illustrate your answer with the help of an example.

Ans: Nested if statements can be useful in Python when you want to check multiple conditions and perform different actions based on the results of those conditions.

Here's an example:
```
x = 10
y = 5
if x > y:
    print("x is greater than y")
    if x > 20:
        print("x is also greater than 20")
    else:
        print("x is not greater than 20")
else:
    print("x is not greater than y")
```

Nested if statements can also be used in more complex programs, where multiple conditions need to be checked in a hierarchical way. However, it's important to keep in mind that too many nested if statements can make code hard to read and maintain, so it's a good idea to use them judiciously and consider other options, such as using elif statements or creating separate functions to handle different conditions.

2) Explain the utility of break and continue statements with the help of an example.
Ans: In Python, break and continue are two control flow statements that allow you to alter the flow of a loop. The break statement is used to terminate the loop prematurely. When break is executed inside a loop, it causes the loop to exit immediately, regardless of whether the loop has completed all its iterations.

Here's an example:

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

In this example, the for loop will iterate from 0 to 9, but when i becomes 5, the break statement is executed and the loop is terminated. As a result, the output of this code will be:

```
0
1
2
3
4
```

On the other hand, the continue statement is used to skip over one iteration of a loop. When continue is executed inside a loop, it causes the loop to skip the rest of the code in that iteration and move on to the next iteration. Here's an example:

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

In this example, the for loop will iterate from 0 to 9, but when i is an even number, the continue statement is executed and the rest of the code in that iteration is skipped. As a result, the output of this code will be:

```
1
3
```

5
7
9
In both cases, the break and continue statements allow you to modify the behavior of a loop to suit your needs.

3) Write a program that accepts a string from the user and calculate the number of digits and letters in the string.

Ans:

```
string = input("Enter a string: ")
num_digits = 0
num_letters = 0
for char in string:
    if char.isdigit():
        num_digits += 1
    elif char.isalpha():
        num_letters += 1
print("Number of digits:", num_digits)
print("Number of letters:", num_letters)
```

Date: ______________ **Signature of faculty in-charge**