EXPERIMENT NO: 1
EXPERIMENT NAME: Study on Arithmetic Operations (Addition, Subtraction, Multiplication, Division) in Python

OBJECTIVE:
To study and implement basic arithmetic operations in Python.

THEORY:
Python provides built-in arithmetic operators to perform mathematical calculations. The main operators are:

- for addition
- for subtraction
- for multiplication
  / for division

These operators work with integers and floating-point numbers. Division (/) always returns a float result.

CODE:

```
a = 20
b = 5

print("addition:", a + b)
print("subtraction:", a - b)
print("multiplication:", a * b)
print("division:", a / b)
```

OUTPUT:
addition: 25
subtraction: 15
multiplication: 100
division: 4.0

CONCLUSION:
Basic arithmetic operations were successfully performed using Python operators.

EXPERIMENT NO: 2
EXPERIMENT NAME: Study on Arithmetic Operations (Subtraction and Multiplication) in Prolog

OBJECTIVE:
To study subtraction and multiplication in Prolog.

THEORY:
Prolog is a logic programming language. Arithmetic expressions are evaluated using the "is" operator. The "is" operator calculates the right-hand expression and assigns the result to a variable.

CODE:

```
subtract(X, Y, R) :- R is X - Y.
multiply(X, Y, R) :- R is X * Y.
```

OUTPUT:

```
?- subtract(10, 4, R).
R = 6

?- multiply(6, 7, R).
R = 42
```

CONCLUSION:
Subtraction and multiplication were successfully implemented in Prolog using the "is" operator.

EXPERIMENT NO: 3
EXPERIMENT NAME: Prolog Program for Addition of Two Numbers

OBJECTIVE:
To write a Prolog program to add two numbers.

THEORY:
In Prolog, addition is performed using the "is" operator. It evaluates the arithmetic expression and returns the result.

CODE:

```
add(X, Y, R) :- R is X + Y.
```

OUTPUT:

```
?- add(8, 9, R).
R = 17
```

CONCLUSION:
The program successfully adds two numbers in Prolog.

EXPERIMENT NO: 4
EXPERIMENT NAME: Prolog Program to Find the Sum of All Numbers in a List

OBJECTIVE:
To write a Prolog program to calculate the sum of all numbers in a given list.

THEORY:
This program uses recursion.
Base case: The sum of an empty list is 0.
Recursive case: Add the head element to the sum of the remaining list.

CODE:

```
sum_list([], 0).
sum_list([H|T], S) :-
    sum_list(T, S1),
    S is H + S1.
```

OUTPUT:

```
?- sum_list([1,2,3,4,5], S).
S = 15
```

CONCLUSION:
The recursive Prolog program successfully calculated the sum of all elements in a list.

EXPERIMENT NO: 5
EXPERIMENT NAME: Python Program to Reverse a Given Number

OBJECTIVE:
To write a Python program to reverse a given number.

THEORY:
The program extracts digits using modulus (%) and builds the reversed number using multiplication and floor division.

CODE:

```
num = 12345
rev = 0

while num > 0:
    rev = rev * 10 + num % 10
    num //= 10

print("reversed number:", rev)
```

OUTPUT:
reversed number: 54321

CONCLUSION:
The number was successfully reversed using arithmetic operations and a loop in Python.

EXPERIMENT NO: 6
EXPERIMENT NAME: FOL to Prolog Conversion (Likes Problem)

OBJECTIVE:
To convert First Order Logic (FOL) statements into Prolog facts and rules.

THEORY:
Facts represent true statements.
Rules define logical relationships between facts.
Queries are used to retrieve information.

CODE:

```
likes(sakib, cricket).
likes(sakib, football).
likes(sakib, rugby).

likes(riad, football).
likes(riad, rugby).

likes(sabbir, flower).
likes(sabbir, custurd).
likes(sabbir, fruits).

likes(sabbir, X) :- likes(_, football), X = football.
likes(sakib, X) :- likes(sabbir, X).
```

OUTPUT:

```
?- likes(sabbir, X).
?- likes(sakib, X).
```

(Results will display matching values of X.)

CONCLUSION:
Logical statements were successfully represented and solved using Prolog.

EXPERIMENT NO: 7
EXPERIMENT NAME: FOL to Prolog Conversion (Family and Gender Problem)

OBJECTIVE:
To represent family relationships and gender using Prolog.

THEORY:
Family relationships are represented as facts.
Queries help retrieve specific information using variables.

CODE:

```
brother(rashid, tamim).
brother(rashid, rishab).

sister(champa, tamim).
sister(champa, rishab).

man(rashid).
man(tamim).
man(rishab).
woman(champa).
```

OUTPUT:

```
?- woman(X).
?- man(X).
?- sister(champa, X).
?- brother(rashid, X).
?- brother(X, tamim).
```

(Prolog returns matching names.)

CONCLUSION:
Family and gender relationships were successfully implemented and queried in Prolog.

EXPERIMENT NO: 8
EXPERIMENT NAME: Implementation of ELIZA Chatbot Using Simple Relational Knowledge

OBJECTIVE:
To implement a simple ELIZA chatbot using Python.

THEORY:
ELIZA is a basic chatbot that responds using pattern matching. It checks user input and provides predefined responses.

CODE:

```
while True:
    user = input("you: ").lower()

    if user == "hi":
        print("eliza: hello")
    elif "sad" in user:
        print("eliza: why are you sad")
    elif "happy" in user:
        print("eliza: that's good to hear")
    elif user == "bye":
        print("eliza: goodbye")
        break
    else:
        print("eliza: tell me more")
```

OUTPUT:
Example interaction:

you: hi
eliza: hello

you: I am sad
eliza: why are you sad

you: bye
eliza: goodbye

CONCLUSION:
A simple ELIZA chatbot was successfully implemented using conditional statements in Python.