EX.NO:8

DATE:22/02/2025

ANALYSE SENTENCE

AIM:

To translate sentences into predicate and first-order logic and analyse the same.

PROCEDURE:

1. Import NLTK
2. Define the line parser .
3. Input the sentences and mention the required translations

CODE :

a) Translate the following sentences into propositional logic and verify that they parse with LogicParser. Provide a key that shows how the propositional variable in your translation corresponds to expressions of English.

```
lp('(exists x. likes(Angus,x) & exists y. likes(y,Julia))')
lp('-exists x. smiles(x,Pat)')
lp('-exists x. (cough(x) | sneeze(x))')
lp('exists x. (asleep(x) & all y. (asleep(y) -> (y = x)))')
```

b) Translate the following sentences into predicate-argument formulas of first-order logic.

```
lp('\\x. all y. love(y,x)')

lp('\\x. all y. (love(y,x) | detested(y,x))')

lp('\\x. (all y. love(y,x) & -exists z. detested(z,x))')
```

c) Translate the following sentences into quantified formulas of first-order logic.

```
lp('(exists x. likes(Angus,x) & exists y. likes(y,Julia))')

lp('-exists x. smiles(x,Pat)')

lp('-exists x. (cough(x) | sneeze(x))')

lp('exists x. (asleep(x) & all y. (asleep(y) -> (y = x)))')

lp('like(Angus,Cyril) & hate(Irene,Cyril)')
```

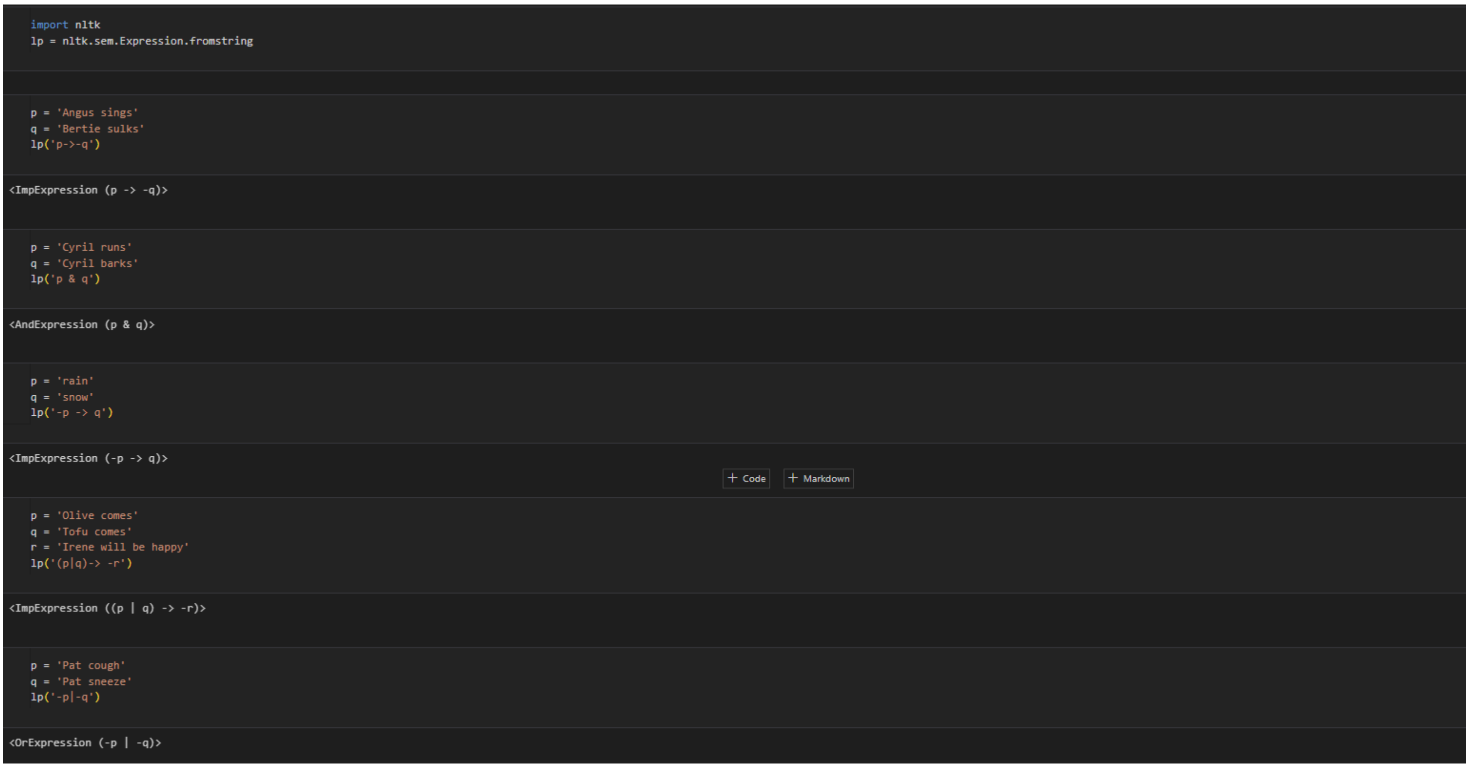
d) Translate the following verb phrases using λ abstracts. quantified formulas of first order logic.

```
lp('\\x. all y. love(y,x)')

lp('\\x. all y. (love(y,x) | detested(y,x))')

lp('\\x. (all y. love(y,x) & -exists z. detested(z,x))')
```

OUTPUT:



```
import nltk
lp = nltk.sem.Expression.fromstring
```

```
p = 'Angus sings'
q = 'Bertie sulks'
lp('p->-q')
```

```
<ImpExpression (p -> -q)>
```

```
p = 'Cyril runs'
q = 'Cyril barks'
lp('p & q')
```

```
<AndExpression (p & q)>
```

```
p = 'rain'
q = 'snow'
lp('-p -> q')
```

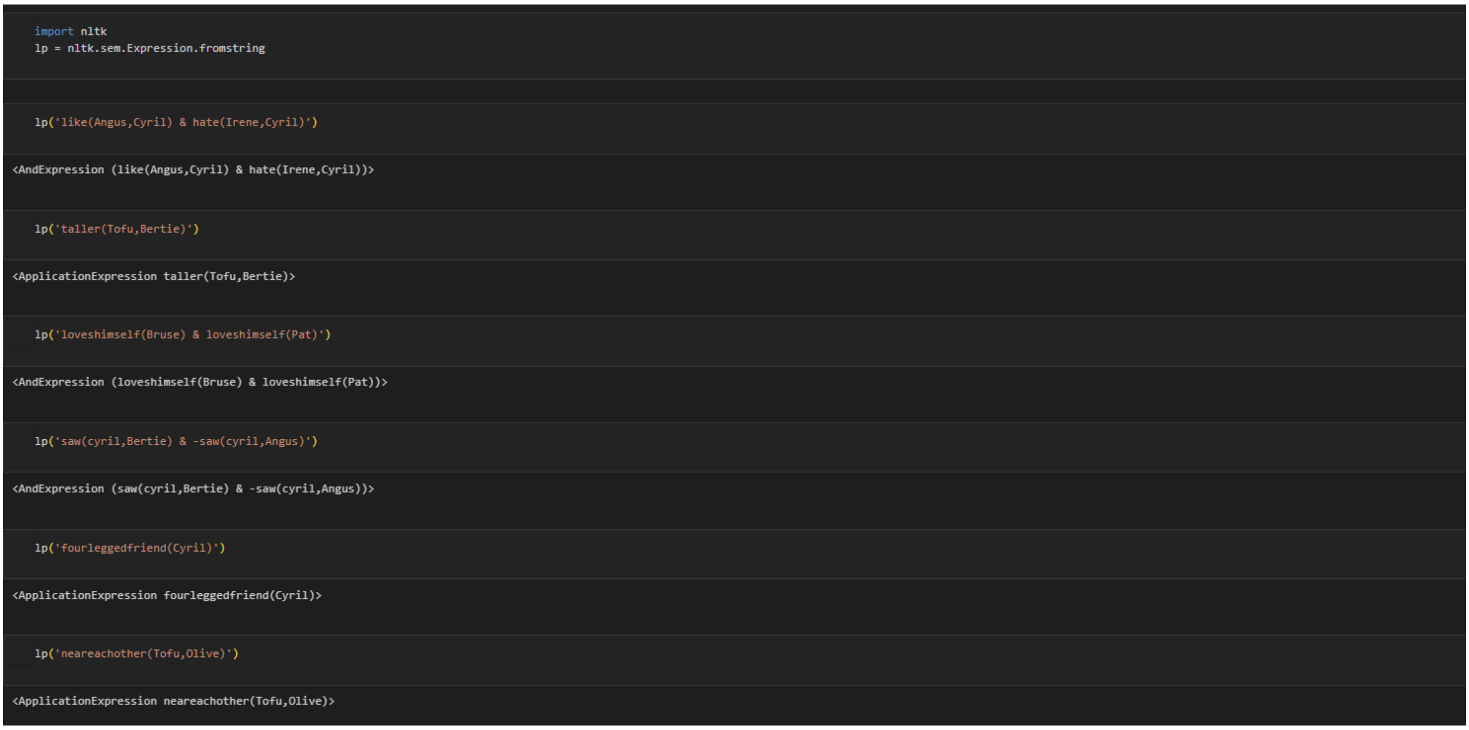
```
<ImpExpression (-p -> q)>
```

```
p = 'Olive comes'
q = 'Tofu comes'
r = 'Irene will be happy'
lp('(p|q)-> -r')
```

```
<ImpExpression ((p | q) -> -r)>
```

```
p = 'Pat cough'
q = 'Pat sneeze'
lp('-p|-q')
```

```
<OrExpression (-p | -q)>
```



```
import nltk
lp = nltk.sem.Expression.fromstring
```

```
lp('like(Angus,Cyril) & hate(Irene,Cyril)')
```

```
<AndExpression (like(Angus,Cyril) & hate(Irene,Cyril))>
```

```
lp('taller(Tofu,Bertie)')
```

```
<ApplicationExpression taller(Tofu,Bertie)>
```

```
lp('loveshimself(Bruse) & loveshimself(Pat)')
```

```
<AndExpression (loveshimself(Bruse) & loveshimself(Pat))>
```

```
lp('saw(cyril,Bertie) & -saw(cyril,Angus)')
```

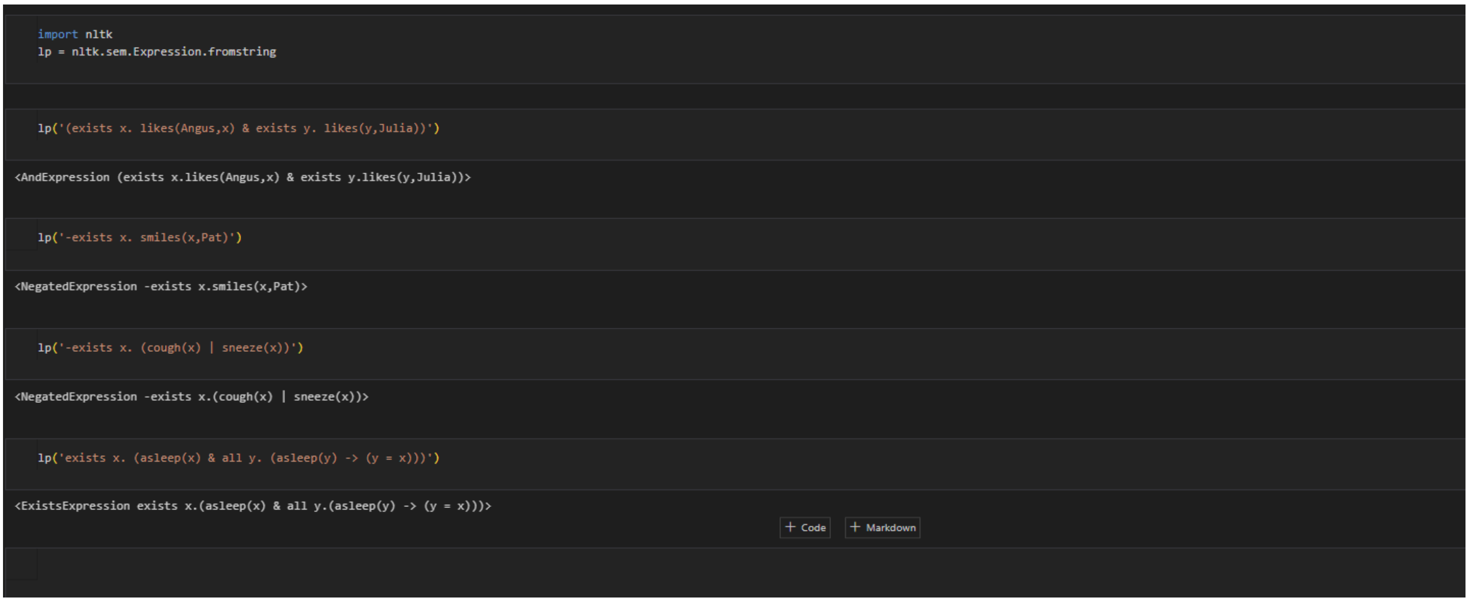
```
<AndExpression (saw(cyril,Bertie) & -saw(cyril,Angus))>
```

```
lp('fourleggedfriend(Cyril)')
```

```
<ApplicationExpression fourleggedfriend(Cyril)>
```

```
lp('neareachother(Tofu,Olive)')
```

```
<ApplicationExpression neareachother(Tofu,Olive)>
```



```
import nltk
lp = nltk.sem.Expression.fromstring
```

```
lp('(exists x. likes(Angus,x) & exists y. likes(y,Julia))')
```

```
<AndExpression (exists x.likes(Angus,x) & exists y.likes(y,Julia))>
```

```
lp('-exists x. smiles(x,Pat)')
```

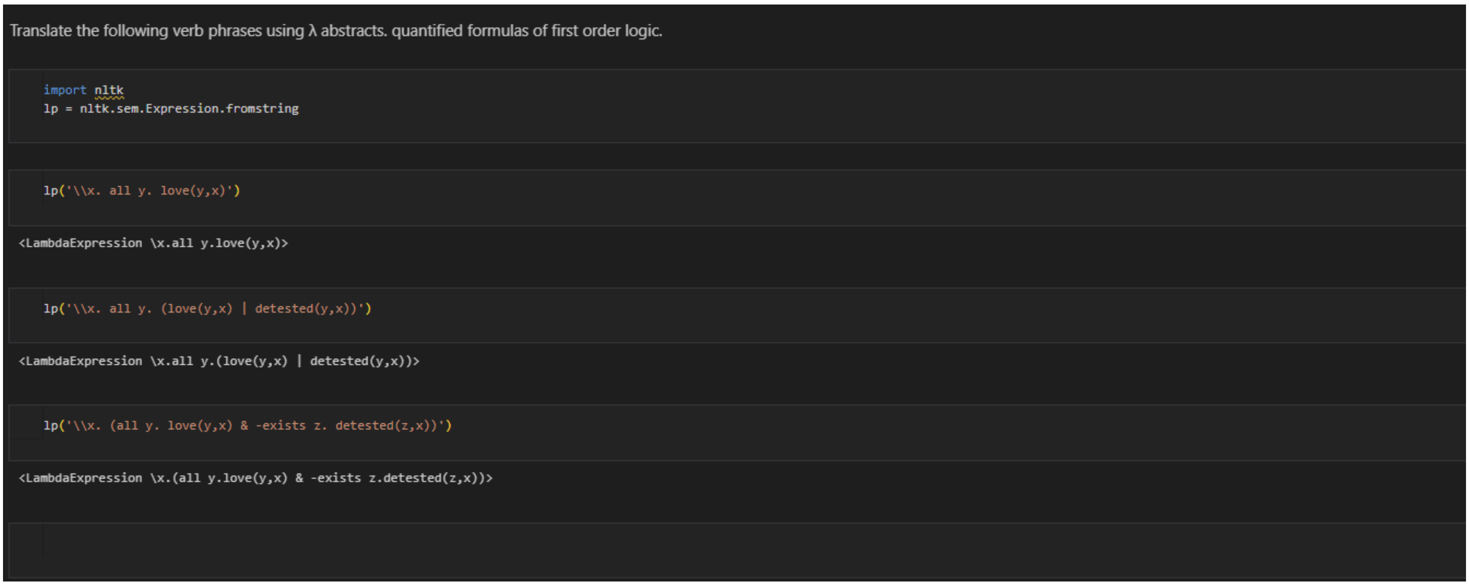
```
<NegatedExpression -exists x.smiles(x,Pat)>
```

```
lp('-exists x. (cough(x) | sneeze(x))')
```

```
<NegatedExpression -exists x.(cough(x) | sneeze(x))>
```

```
lp('exists x. (asleep(x) & all y. (asleep(y) -> (y = x)))')
```

```
<ExistsExpression exists x.(asleep(x) & all y.(asleep(y) -> (y = x)))>
```



Translate the following verb phrases using λ abstracts. quantified formulas of first order logic.

```
import nltk
lp = nltk.sem.Expression.fromstring
```

```
lp('\\x. all y. love(y,x)')
```

```
<LambdaExpression \x.all y.love(y,x)>
```

```
lp('\\x. all y. (love(y,x) | detested(y,x))')
```

```
<LambdaExpression \x.all y.(love(y,x) | detested(y,x))>
```

```
lp('\\x. (all y. love(y,x) & -exists z. detested(z,x))')
```

```
<LambdaExpression \x.(all y.love(y,x) & -exists z.detested(z,x))>
```