7.IMPLEMENTING FOL

```
import re

class FOLConverter:
    def __init__(self):
        self.constants = {}
        self.predicates = {}

    def define_predicates(self, name, arity):
        self.predicates[name] = arity

    def define_constants(self, name):
        self.constants[name] = name

    def translate(self, sentence):
        sentence = sentence.lower().strip()
        if self.match_universal(sentence):
            return self.handle_universal(sentence)
        elif self.match_existential(sentence):
            return self.handle_existential(sentence)
        elif self.match_implication(sentence):
            return self.handle_implication(sentence)
        else:
            return self.handle_basic(sentence)

    def match_universal(self, sentence):
        return re.match(r"(every|all)\s+\w+\s+is\s+\w+", sentence)

    def handle_universal(self, sentence):
        match = re.match(r"(every|all)\s+(\w+)\s+is\s+(\w+)", sentence)
        if match:
            subject = match.group(2)
            predicate = match.group(3)
            return f"∀x ({subject}(x) → {predicate}(x))"
        return sentence

    def match_existential(self, sentence):
        return re.match(r"there\s+is\s+(a|someone)\s+who\s+\w+", sentence)
```

```
    def handle_existential(self, sentence):
        match =
re.match(r"there\s+is\s+(a|someone)\s+who\s+(\w+)\s+(\w+)", sentence)
        if match:
            subject = match.group(1)
            predicate = match.group(2)
            object = match.group(3)
            return f"∃x {predicate}(x, {object})"
        return sentence

    def match_implication(self, sentence):
        return re.match(r"if\s+.*\s+then\s+.*", sentence)

    def handle_implication(self, sentence):
        match = re.match(r"if\s+(.*)\s+then\s+(.*)", sentence)
        if match:
            premise = match.group(1)
            conclusion = match.group(2)
            return f"{premise} → {conclusion}"
        return sentence

    def handle_basic(self, sentence):
        # Handle both "is" (identity) and "is a" (classification)
        match = re.match(r"(\w+)\s+is\s+(\w+)", sentence)
        if match:
            subject = match.group(1)
            predicate = match.group(2)
            return f"{predicate}({subject})"
        return sentence

# Example usage
fol_converter = FOLConverter()
fol_converter.define_predicates("human", 1)
fol_converter.define_predicates("mortal", 1)
fol_converter.define_predicates("loves", 2)

sentences = [
    "John is a human",
    "Every human is mortal",
    "John loves Mary",
```

```
    "There is someone who loves Mary",
    "If it is raining, then the ground is wet"
]

for sentence in sentences:
    fol = fol_converter.translate(sentence)
    print(f"Original: {sentence}")
    print(f"FOL: {fol}\n")
```

Output:

```
Original: John is a human
FOL: a(john)

Original: Every human is mortal
FOL: ∀x (human(x) → mortal(x))

Original: John loves Mary
FOL: john loves mary

Original: There is someone who loves Mary
FOL: ∃x loves(x, mary)

Original: If it is raining, then the ground is wet
FOL: it is raining, → the ground is wet
```

19/ Tuesday

Bafna Gold
Date: Page:

Lab - 7

FOL

Algorithm:

```
Function convertToFOL(sentence):
    Sentence = NormalizeSentence(sentence)
    If IsUniversalQuantification(Sentence):
    Return HandleUniversalQuantification(sentence)
    Else if IsExistentialQuantification(sentence):
    Return HandleExistentialQuantification(sentence)
    Else if IsImplication(sentence)
    Else:
     Return HandleBasicPredicate(Sentence)

End Function

Function NormalizeSentence(Sentence):
Return ToLowerCase(RemovePunctuation(ToLowerWhitespace
                                     (sentence)))
Function IsUniversalQuantification(sentence):
Return "MatchesPattern(sentence, "Every <subject> is
    <predicate>")
Function HandleUniversalQuantification(sentence):
 Subject = Extract Subject(sentence)
 Predicate = Extract Predicate(sentence)
 Return "∀x(" + Subject + "(x) →" + predicate + "(x))"
Function IsExistentialQuantification(sentence):
Return Matchespattern(sentence, "There is someone
 who <predicate> <object>")
Function HandleExistentialQuantification(sentence):
 predicate = Extractpredicate(sentence)
  object = Extract object(sentence)
 Return "∃x" + predicate + "(x," + object + ")"
```

```
Function IsImplication (sentence):
Return MatchesPattern (sentence, "If <condition>
    then <consequence>")
Function HandleImplication (sentence):
  Condition = ExtractCondition (sentence)
  Consequence = ExtractConsequence (sentence)
  Return condition + "→" + Consequence
Function HandleBasePredicate (sentence):
  subject = ExtractSubject (sentence)
  Predicate = ExtractPredicate (sentence)
  Return Predicate + "(" + Subject + ")"
Function ExtractsMatched pattern (sentence, pattern):
Return True if Sentence matched pattern else False
Function ExtractSubject (sentence):
Return ExtractWordBasedonPosition (sentence, 1)
Function Extract object (sentence):
Return ExtractWordBasedonPosition (sentence, 3)
Function ExtractCondition (sentence):
Return ExtractWordBaseBefore ("then", Sentence)
Function ExtractConsequence (sentence):
Return ExtractWordAfter ("then", Sentence)
Function ExtractwordBasedonPosition (sentence, position)
  words = SplitSentenceIntowords (sentence)
  Return words[position]

Function SplitSentenceIntowords (sentence)
    Return sentence.split (" ")
```

Output:

original : John is a human
FOL : a (John)

original : Every human is mortal
FOL : $\forall x$ (human (x) → mortal (x))

original : John loves mary
FOL : John loves mary
original : There is someone who loves mary
FOL : $\exists x$ Loves (x, mary)

original : If it is raining, then the ground is wet
FOL : it is raining, → the ground is wet