Republic of the Philippines
Laguna State Polytechnic University
Province of Laguna

ISO 9001:2015 Certified
Level I Institutionally Accredited

LSPU Self-Paced Learning Module (SLM)

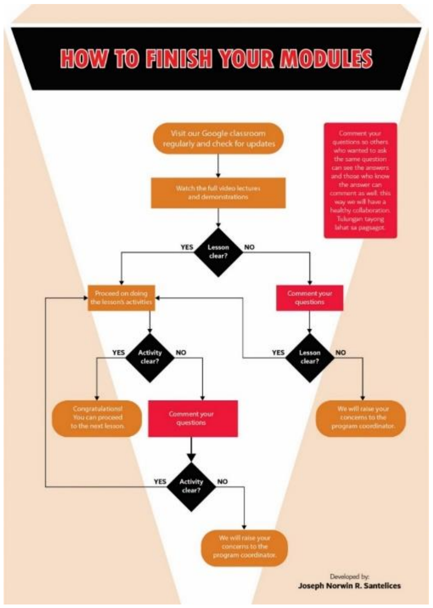
Course	**Bachelor of Science in Computer Science**
Sem/AY	**First Semester/2023-2024**
Module No.	**4**
Lesson Title	**Logical Programming**
Week Duration	**4**
Date	
Description of the Lesson	This lesson will discuss the Overview and function of Logical Programming in Python.

Learning Outcomes

Intended Learning Outcomes	Students should be able to meet the following intended learning outcomes: • Describe Logical Programming • Explain the basic features of Logical Programming • Classify its examples • Apply Python relation, facts, rules.
Targets/ Objectives	At the end of the lesson, students should be able to learn, practice and apply the ff: • *Logical Programming in Python* • *Uses Cases using Logical Programming* • *Solving puzzles using Artificial Intelligence* • *Building the Logic*

Student Learning Strategies

Online Activities (Synchronous/	A. Online Discussion via Google Classroom. You will be directed to join in the online classroom where you will have access to the video lectures, course materials, and evaluation tools 24/7.

Asynchronous)	To have access on the Online Lectures, refer to this link: ___________________. To ensure a healthy collaboration, you may post your inquiries on a particular topic and the instructor including all the members of the class can acknowledge the inquiry. (For further instructions, refer to your Google Classroom and see the schedule of activities for this module) B. Learning Guide Questions: Refer to the following diagram as your guide through the course.  HOW TO FINISH YOUR MODULES Visit our Google classroom regularly and check for updates Watch the full video lectures and demonstrations Lesson clear? YES NO Proceed on doing the lesson's activities Comment your questions Activity clear? YES NO Lesson clear? YES NO Congratulations! You can proceed to the next lesson. Comment your questions We will raise your concerns to the program coordinator. Activity clear? YES NO We will raise your concerns to the program coordinator. Comment your questions so others who wanted to ask the same question can see the answers and those who know the answer can comment as well. this way we will have a healthy collaboration. Tulungan tayong lahat sa pagsagot. Developed by: Joseph Norwin R. Santelices ***Note:*** *The insight that you will post on online discussion forum using Learning Management System (LMS) will receive additional scores in class participation*
Offline Activities (e-Learning/Self-Paced)	**Lecture Guide** **Logical Programming** The relationship between mathematical logic and computer science runs deep. As such, there are a number of programming language tools that enable you to rapidly code

efficient logic systems for deployment in a variety of high-tech arenas, such as Artificial Intelligence, Textual Analysis, and so on.

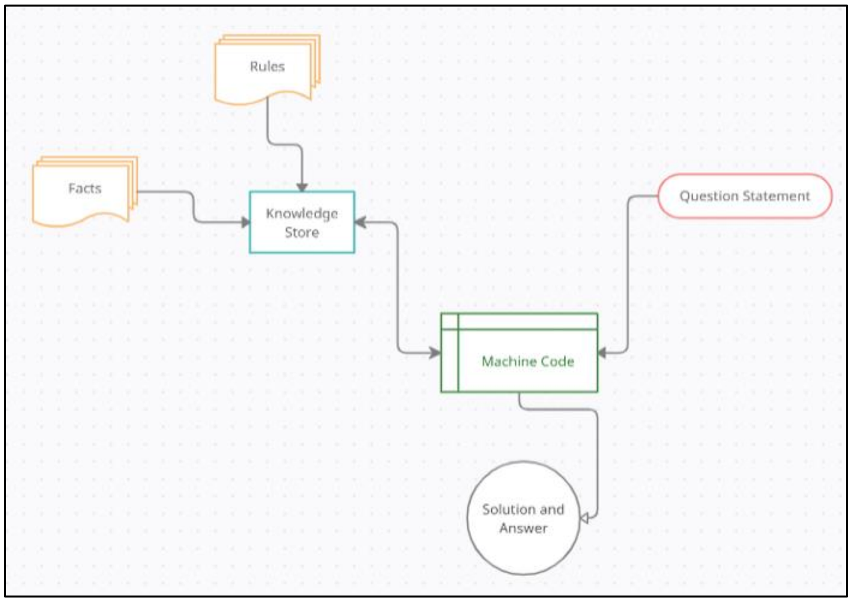
In this unit, you will learn about Logical Programming (also known as Declarative Programming) and why you would use it to solve these sorts of problems. We will discuss the fundamental features of Logical Programming and identify what distinguishes Logical Programming from other programming paradigms. By the end of this unit, you will be able to identify the problem domain that Logical Programming covers and recognize when you should take advantage of Logical Programming's considerable power.

What is Logic Programming in Python?

Python Logic programming is a programming paradigm that sees computation as automatic reasoning over a database of knowledge made of facts and rules. It is a way of programming and is based on formal logic.

Idea Description

As the words suggest, Logic Programming is combined of two distinct ideas — logic and programming. The logic here details the facts and rules that the programming structure needs to understand. A logical approach to programming requires a set of input rules that the code learns and then infers an output on a new related fact it has not seen before. This can also be viewed as a form of a learning algorithm with explicit instruction of understanding.



Representation of the flow of Logic Programming code | Image by Author

Writing code in Python, C, C++, Java, etc. we have observed paradigms like object-oriented programming (OOPS), abstraction, looping constructs, and numerous other states of programming. Logic Programming is just another programming paradigm that works on relationships. These relationships are built using facts and rules and are stored as a database of relations. It is a programming methodology that works on formal and explicit logic of events.

Relation: Relations are the basis of logic programming. A relation can be defined as a fact that follows a certain rule. For example, a relation given by [A ⍰ B] is read as "if A is true, then B occurs". In language terms, this can be read as, "If you are an Engineer, then you are a Graduate" and infers that, "Engineers are Graduates". In programming languages, the semantics of writing relations change based on the language's syntax, but this is the overall rationality behind what relations mean.

Facts: Every program that is built on logic needs facts. To achieve a defined goal, facts need to be provided to the program. As the name suggests in general English, facts are merely the truth. True statements that represent the program and the data. For instance, Washington is the capital of the United States of America.

Rules: Rules, like programming syntax, are the constraints that help in drawing conclusions from a domain. These are logical clauses that the program or the fact needs to follow to build a relation. You can think of it like this, the fact is that Raman is a man. Now, gender can be a singular entity, that is a rule. A man cannot be a woman. Therefore, the relations we build here are that, since Raman is a man, he cannot be a woman. This is how rules are built:

For example:
predecessor(A,B) :- parent(A,B).
predecessor(A,C) :- parent(A,B), predecessor(B,C).

This can be read as, for every A and B, if A is the parent of B and B is a predecessor of C, A is the predecessor of C. For every A and B, A is the predecessor of C, if A is the parent of B and B is a predecessor of C.

Building the Logic

Since we are focusing on creating applications solely based on Logic, it is necessary to first understand how this logic works and how it can be derived. Below we will follow the hierarchy of building "Logic".

Propositional Logic: All forms of code building start with the most basic form of logic, which is propositional. Here, statements are made by propositions, that can be defined as declarative statements having either of two outcomes; True or False. It is a knowledge representation technique used in Mathematics.

A proposition formula that is always true is called Tautology, and it is also called a valid sentence. A proposition formula that is always false is called **Contradiction**.

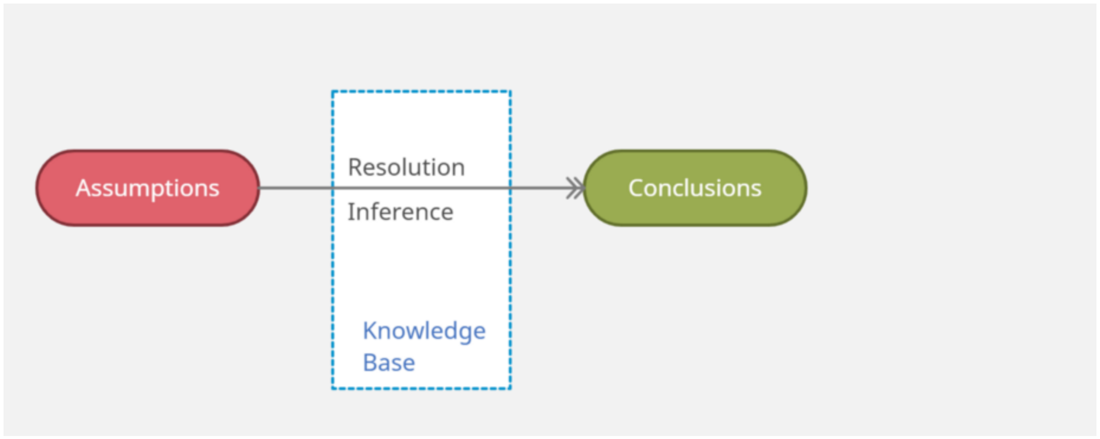
It is Sunday. (True)
The Sun rises from North (False proposition)
3+6= 7 (False proposition)
3 is a prime number. (True)

First-Order Logic: This is an extension of the Propositional Logic. First-order also builds a boolean representation, but it does it in a space called the domain. This domain is a partial view of the world with a limited environment. First-Order logic is made up of syntax and semantics.

In addition to facts (as assumed by propositional logic), First-Order logic also assumes that the world contains objects, relations, and functions that represent occurrences which cannot be classified as true or false.

For example, consider the sentence "X is an Integer". This sentence consists of two parts. The second part *"is an integer"* is a Fact. A variable can either be an integer or not (Boolean, hence, Propositional). But, the first part "X", also called the subject, is not Boolean. It is a naturally occurring object and could have been any variable. The combination of a natural substance (non-boolean) with propositional logic is the First-Order Logic. These are a few examples of these non-boolean substances of First-Order Logic.
a. **Constants (**1, 2, A, John, etc.)
b. **Variables (**x, y, z, a, b)
c. **Predicates (**Brother, Father, >,<)
d. **Function (**sqrt, LeftLegOf, etc.)
e. **Connectives (**∧, ∨, ¬, ⇒, ⇔)
f. **Equality (**==)

Clausal-Form Logic: The final form of logic is clausal form. This is used in Logic Programming. Clausal-Form logic is a subset of First-Order Logic. The knowledge given to a first-order logic system is manipulated using the resolution inference system that enables the proof of theorems.



Using the Resolution-Inference method to extract Propositions from the Knowledge base and build Logic | Image by Author

Use Cases of Logic Programming

Logic Programming is extensively used in Natural Language Processing (NLP) since understanding languages is about the recognition of patterns that numbers cannot represent.

It is also used in prototyping models. Since expressions and patterns can be replicated using logic, prototyping is made easy.

Pattern matching algorithms within image processing, speech recognition, and various other cognitive services also use logic programming for pattern recognition.

Scheduling and Resource Allocation are major operations tasks that logic programming can help solve efficiently and completely.

Mathematical proofs are also easy to decode using logic programming.

Solving Puzzles using Artificial Intelligence

Logic Programming can be used to solve numerous mathematical problems that will ultimately help in building an artificially intelligent machine. In the sections coming next, we will observe how Logic Programming can be used to evaluate expressions in mathematics, make programs learn operations, and form predictions. We will also solve a real problem using two libraries that influence logic programming in Python.

Kanren: Kanren is a library within PyPi that simplifies ways of making business logic out of code. The logic, rules, and facts we discussed previously can be turned into code using 'kanren'. It uses advanced forms of pattern matching to understand the input expressions and build its own logic from the given input. We will be using this library in the sections below for mathematical computations. The import and installation steps are mentioned in the code section that follows.

SymPy: SymPy stands for symbolic computation in Python and is an open-sourced library. It is used for calculating mathematical constructs using symbols. The SymPy project aims to establish a completely featured Computer Algebra System (CAS). The aim here is to keep the understanding of the code simple and comprehensive.

Evaluating Mathematical Idioms using Logic Programming

Algorithms are nothing but implementation of logic and control. Similarly, when the logic runs a mathematical function, we call it a mathematical expression. These expressions are the inputs we give to the program, based on which the program understands the rules that are present in the logic. Based on the understanding of these rules, future expressions can also be evaluated. Let us see an implementation of Logic Programming to evaluate mathematical expressions:

CODE 1: Check for Mathematical Patterns
```
pip install kanren
pip install sympy# Define values that will undertake the addition and
multiplication operations
addition = 'add'
multiplication = 'mul'# Define facts and properties of each operation
fact(commutative, multiplication)
fact(commutative, addition)
fact(associative, multiplication)
fact(associative, addition)# Declare the variables that are going to form the
expression
var_x, var_y, var_z = var('var_x'), var('var_y'), var('var_z')# Build the
correct pattern that the program needs to learnmatch_pattern = (addition,
(multiplication, 4, var_x, var_y), var_y, (multiplication, 6,
var_z))match_pattern = (addition, (multiplication, 3, 4), (multiplication,
(addition, 1, (multiplication, 2, 4)),2))# Build 3 distinct expressions to
test if the function has learnttest_expression_one = (addition,
(multiplication, (addition, 1 , (multiplication, 2, var_x )),
var_y) ,(multiplication, 3, var_z ))test_expression_two = (addition,
(multiplication, var_z, 3), (multiplication, var_y, (addition,
(multiplication, 2, var_x), 1)))test_expression_three = (addition  ,
(addition, (multiplication, (multiplication, 2, var_x), var_y), var_y),
(multiplication, 3, var_z))# Test the evaluations of the expression on the
test expressions
run(0,(var_x,var_y,var_z),eq(test_expression_one,match_pattern))
>>> ((4, 2,
4),)run(0,(var_x,var_y,var_z),eq(test_expression_two,match_pattern))
>>> ((4, 2,
4),)run(0,(var_x,var_y,var_z),eq(test_expression_three,match_pattern))
>>> ()# Since the first two expressions satisfy the expression above, they
return the values of individual variables. The third expression is
structurally different and therefore does not match.CODE 2: Symbolic
Representations of Logic# Running Mathematical Evaluations using SymPy
print (math.sqrt(8))
>>> 2.8284271247461903# Although the Math Square Root function gives an output
for the Square Root of 8.
# We know this is not accurate since the square root of 8 is a recursive, non-
ending real numberprint (sympy.sqrt(3))
>>> sqrt(3)# Sympy on the other hand, symbolizes the output and shows it as
root of 3
# In case of actual square roots like 9, SymPy gives the correct result and
not a symbolic answerCODE 3: Prime Number check using Logic Programming#
Defining a function to build the expression for Prime Number check
def exp_prime (input_num):
if isvar(input_num):
  return condeseq([(eq, input_num, x)] for x in map(prime,
iter_one.count(1)))
else:
  return success if isprime (input_num) else fail# Variable to use
n_test = var()set(run(0, n_test,(membero,
n_test,(12,14,15,19,21,20,22,29,23,30,41,44,62,52,65,85)),( exp_prime,
n_test)))
>>> {19, 23, 29, 41}run(7, n_test, exp_prime( n_test ) )
>>> (2, 3, 5, 7, 11, 13, 17)
```

The code mentioned above only consists of all necessary pseudo logic and will not run independently on any IDE. The complete code is present in the repository linked below and I would encourage you to go through the Google Colab notebook from the repository that contains the entire working code.

Performance Tasks

CMSC308 Programming Languages

MODULE 4
Logical Programming

Activity 4: Logical Programming using Python

Write a program equation/1 that solves a linear one variable equation. The equation consistst of one variable, two numbers, one equality and one operator. The operator is either addition, subtraction, multiplication or division. For example

```
?- equation(X + 2 = 5).
X = 3.
?- equation(2 - 3.5 = Dif).
Dif = -1.5.
```

Instruction:

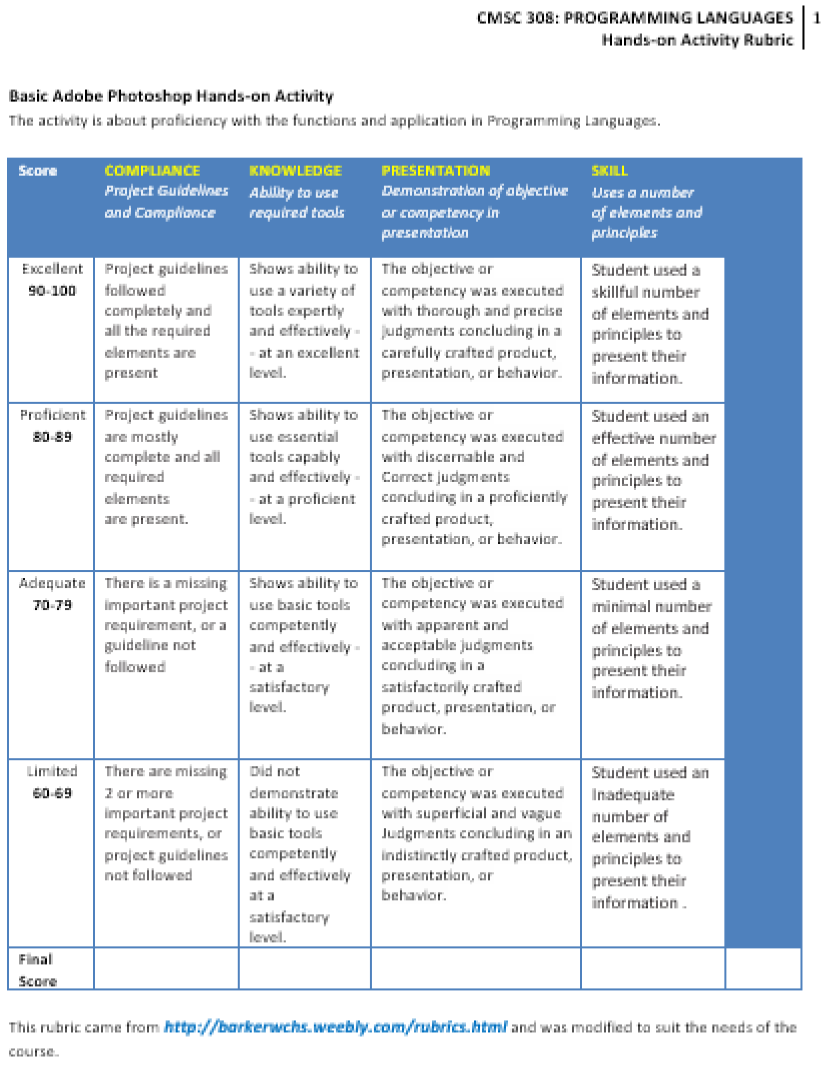
File name format should be:

CMSC308m4_lastname_firstname_year&section

You can submit it anytime before the end of the semester but take note, i can see the date of your submission and any of your activities in our group on my end. Please be proactive. Keep safe and good luck.

Rubric:
Hands-On Activity rubric

Understanding Directed Assess



Basic Adobe Photoshop Hands-on Activity

The activity is about proficiency with the functions and application in Programming Languages.

Score	COMPLIANCE *Project Guidelines and Compliance*	KNOWLEDGE *Ability to use required tools*	PRESENTATION *Demonstration of objective or competency in presentation*	SKILL *Uses a number of elements and principles*	
Excellent **90-100**	Project guidelines followed completely and all the required elements are present	Shows ability to use a variety of tools expertly and effectively -- at an excellent level.	The objective or competency was executed with thorough and precise judgments concluding in a carefully crafted product, presentation, or behavior.	Student used a skillful number of elements and principles to present their information.	
Proficient **80-89**	Project guidelines are mostly complete and all required elements are present.	Shows ability to use essential tools capably and effectively -- at a proficient level.	The objective or competency was executed with discernable and Correct judgments concluding in a proficiently crafted product, presentation, or behavior.	Student used an effective number of elements and principles to present their information.	
Adequate **70-79**	There is a missing important project requirement, or a guideline not followed	Shows ability to use basic tools competently and effectively -- at a satisfactory level.	The objective or competency was executed with apparent and acceptable judgments concluding in a satisfactorily crafted product, presentation, or behavior.	Student used a minimal number of elements and principles to present their information.	
Limited **60-69**	There are missing 2 or more important project requirements, or project guidelines not followed	Did not demonstrate ability to use basic tools competently and effectively at a satisfactory level.	The objective or competency was executed with superficial and vague Judgments concluding in an indistinctly crafted product, presentation, or behavior.	Student used an Inadequate number of elements and principles to present their information .	
Final Score					

This rubric came from ***http://barkerwchs.weebly.com/rubrics.html*** and was modified to suit the needs of the course.

Learning Resources

Books:

- CS404-Archive_ Unit 7 Learning Outcomes _ Saylor Academy

Website:

- *https://towardsdatascience.com/logic-programming-and-the-design-of-humanistic-ai-using-python-6ddb7019caa2*
- https://learn.saylor.org/course/view.php?id=79&sectionid=786
- https://www.youtube.com/watch?v=cKaEI73nO1M&t=15s
- https://www.youtube.com/watch?v=6aaXKOtjEr8