1. Symbols can now be manipulated using some of python operators using _____.

a)+. b) && c) ? d)$

2. SymPy is able to solve algebraic equations, in one and several variables using

a) solveset() b) series() c) limit() d)real()

3. Finite state machines are used for____

a)Pseudo random test patterns

b)Deterministic test patterns

c)Random test patterns

d)Algorithmic test patterns

4. Identify the latest version of wxPython that supports both Python 2 and Python 3.

a)wxPython Phoenix b)Phoenix c) wxJython d)Sphinx

5. Identify the method that organizes the widgets in blocks before placing in the parent widget.

a)loop() b) Pack() c) mainloop() d)Tk()

6. Which mechanism provides control to the program.

a)Unifier b)Automatic backtracking c)substitution d)declarative semantic

7. Logic programming language is a_____ .

a)non-procedural b) predicate c)declarative semantics d) clauses

8. Which of the following is correct for predicate “Some boys play football”?

a) ∃x boys(x) → play(x, football)

b) ∃ boys(x) → play(x, football)

c) ∀x boys(x) → play(x, football)

d) ∃x ∀boys(x) → play(x, football)

9. Which of the following programming paradigms allow us to write programs and know they are correct before running them?

a) Automata based Programming Paradigm

b) Logical Programming Paradigm

c) Dependent type Programming Paradigm

d) Imperative Programming Paradigm

10. _____ are file descriptors, which happen to be connected to network sources of data rather than to data stored on a filesystem.

a)IP Address b)IPV4 c)IPV2 d)Sockets

11. Find the output of the following program

```
from sympy import solve
 x = Symbol('x')
 expr = x**2 + 5*x + 4
solve(expr, dict=True)
```

a) [{x: -4}, {x: -1}]

b) [{x: -6}, {x: -1}]

c) [{x: -1}, {x: -4}]

d) [{x: 4}, {x: -1}]

13. Find the output of the following program

```
from sympy import *
x = symbols('x')
expr = sin(x)/x;
print("Expression : {}".format(expr))
 # Use sympy.limit() method
limit_expr = limit(expr, x, 0)
print("Limit of the expression tends to 0 : {}".format(limit_expr))
```

a) Expression : cos(x)/x

Limit of the expression tends to 0 : 2

b) Expression : tan(x)/x

Limit of the expression tends to 0 : 3

c) Expression : sin(x)/x

Limit of the expression tends to 1 : 0

d) Expression : sin(x)/x

Limit of the expression tends to 0 : 1

14. Identify the single graph node which can be connected to it's execution counterpart.

a) Machine b) Automata c) State d) Transitions

15. Minimum number of argument we pass in a function to create a rectangle using canvas tkinter?

a) 2 b) 4 c) 6 d) 5

16. Essential thing to create a window screen using tkinter python?

a) call tk() function b) create a button c) To define a geometry d) All of the above

17. _____ is an application-level block of transmitted data.

a)data b)datagram c)segmentation d)Fragmentation

18. _____refers to the address family ipv4.

a)Af_INET b) AS_INET c) AG_INET

d) AN_INET

19. Let x be a variable which refers to Universe of Disclosure such as x1,x2....xn then ,how to represent this statement using quantifiers " All Man working in Industry"

a) ∀x man(x) → work (x, Industry)

b) ∀x man(x) → work(industry).

c) ∀x → work (x, industry)

d) ∀x man(x) → Industry(work)

20. How do you represent the statement "Some boys play cricket "

a) ∃x boys(x) → play(all)

b) ∀x boys(x) → play(x, cricket)

c) ∃x boys(x) → play(x, cricket)

d)∃x^ ∀x boys(x) → play(x, cricket)

21. The correct order of methods used in server socket is

a) Socket(), Bind(), listen(), accept()

b) Socket(), listen(), bind(), accept()

c) Socket(), accept(), bind(), listen()

d) Socket(), bind(), accept(), listen()

Part B

1. Write a program using factor, satisfiable, solve methods in sympy python.

```
import sympy as sp

# Factor a polynomial expression
expr = sp.factor('x**2 + 2*x + 1')
print(expr)

# Check satisfiability of a boolean expression
p, q = sp.symbols('p q')
expr = p & (~q | p)
print(sp.satisfiable(expr))

# Solve an equation
x = sp.symbols('x')
expr = sp.Eq(x**2 + 2*x - 3, 0)
solution = sp.solve(expr, x)
print(solution)
```

Output:

```
(x + 1)**2
{q: False, p: True}
[-3, 1]
```

2. Discuss about dependent functions and dependent pairs.

Dependent functions and dependent pairs are two concepts from dependent type theory, which is a branch of mathematical logic and computer science.

Dependent functions:

In dependent type theory, a function can have a type that depends on a value of another type. This means that the type of the output of the function can vary depending on the input value. Such a function is called a dependent function or a dependent type. In other words, dependent functions are functions that map elements of one type to elements of another type, where the type of the output depends on the value of the input.

```
from typing import TypeVar

T = TypeVar('T')

def head(lst: list[T]) -> T:
   return lst[0]
```

In this example, the function head takes a list of some type T and returns an element of type T. The type of the output depends on the type of the input.

Dependent pairs:

In dependent type theory, a dependent pair is a pair of values where the type of the second value depends on the value of the first value. Such a pair is also called a sigma type or a

dependent sum type. In other words, dependent pairs are pairs of values where the type of the second value varies depending on the value of the first value.

```
from typing import Tuple

def get_name_and_age() -> Tuple[str, int]:
    name = input("Enter your name: ")
    age = int(input("Enter your age: "))
    return (name, age)
```

In this example, the function get_name_and_age returns a pair of values: a string (the person's name) and an integer (the person's age). The type of the second value (the age) depends on the value of the first value (the name). If the name is not a string, then the return type of the function is not well-defined

3. Explain about logic quantifiers such as For all and there exists and write a logic program for it using python. Use Kanren library.

In logic, quantifiers are used to specify the quantity of elements that satisfy a given proposition or condition. Two common quantifiers are "for all" (denoted by the symbol ∀) and "there exists" (denoted by the symbol ∃).

The "for all" quantifier (∀) indicates that a given proposition is true for every element in a set. For example, ∀x P(x) means that proposition P is true for every element x in the set.

The "there exists" quantifier (∃) indicates that there is at least one element in a set that satisfies a given proposition. For example, ∃x P(x) means that there exists at least one element x in the set for which proposition P is true.

```
from kanren import *
from kanren.core import success, fail, eq, condeseq
from kanren.facts import Relation, facts

# Define a Relation for our predicates
parent = Relation()
ancestor = Relation()

# Define some facts about the parent relation
facts(parent, ("Bob", "Alice"), ("Alice", "Carol"), ("Alice", "David"), ("David", "Emily"))

# Define the forall quantifier
def forall(var, clause):
    return condeseq([clause(var, val) for val in run(0, var)])

# Define the exists quantifier
def exists(var, clause):
    return condeseq([clause(var, val) for val in run(1, var)])

# Define a rule to check if X is a parent of Y
```

```
def is_parent(X, Y):
    return membero((X, Y), parent)

# Define a rule to check if X is an ancestor of Y
def is_ancestor(X, Y):
    # Base case: X is a parent of Y
    if is_parent(X, Y):
        return True
    # Recursive case: X is an ancestor of Z and Z is an ancestor of Y
    else:
        return exists(lambda Z: conde((is_parent(X, Z), is_ancestor(Z, Y))))

# Test the rules
assert run(0, "X", is_parent("Bob", "X")) == ("Alice",)
assert run(0, "X", is_parent("Alice", "X")) == ("Carol", "David")
assert run(0, "X", is_ancestor("Bob", "X")) == ("Alice", "Carol", "David", "Emily")
assert run(0, "X", forall("Y", lambda Y: is_parent(Y, "X"))) == ("Alice", "Bob")
assert run(0, "X", forall("Y", lambda Y: is_ancestor(Y, "X"))) == ("Bob",)
```

4. Write a program using GUI tkinter python to implement a Menu widget.

```
import tkinter as tk

def do_nothing():
    pass

root = tk.Tk()

# Create a menu bar
menu_bar = tk.Menu(root)
root.config(menu=menu_bar)

# Create a File menu and add it to the menu bar
file_menu = tk.Menu(menu_bar)
menu_bar.add_cascade(label="File", menu=file_menu)

# Add some commands to the File menu
file_menu.add_command(label="New", command=do_nothing)
file_menu.add_command(label="Open", command=do_nothing)
file_menu.add_separator()
file_menu.add_command(label="Exit", command=root.quit)

# Create an Edit menu and add it to the menu bar
edit_menu = tk.Menu(menu_bar)
menu_bar.add_cascade(label="Edit", menu=edit_menu)
```

```
# Add some commands to the Edit menu
edit_menu.add_command(label="Cut", command=do_nothing)
edit_menu.add_command(label="Copy", command=do_nothing)
edit_menu.add_command(label="Paste", command=do_nothing)

# Create a Help menu and add it to the menu bar
help_menu = tk.Menu(menu_bar)
menu_bar.add_cascade(label="Help", menu=help_menu)

# Add a command to the Help menu
help_menu.add_command(label="About", command=do_nothing)

# Start the main loop
root.mainloop()
```

5. Explain in detail about socket and different types of socket.

   A socket is a software interface that allows different processes running on a networked computer to communicate with each other, either on the same machine or across different machines connected to the network. It acts as a bridge between an application and the network, providing a mechanism for data exchange between them.

   There are several types of sockets available, each designed for a specific purpose. Here are the most commonly used types:

   Stream Sockets: Stream sockets provide a reliable, connection-oriented communication channel between two endpoints. They use the Transmission Control Protocol (TCP) as the underlying transport protocol and ensure that data is transmitted in the correct order and without errors. Stream sockets are commonly used for applications that require reliable, error-free data transfer, such as file transfer, email, and web browsing.

   Datagram Sockets: Datagram sockets provide an unreliable, connectionless communication channel between two endpoints. They use the User Datagram Protocol (UDP) as the underlying transport protocol and do not guarantee that data will be delivered in the correct order or without errors. Datagram sockets are commonly used for applications that require fast, low-latency data transfer, such as online gaming and real-time multimedia streaming.

   Raw Sockets: Raw sockets provide direct access to the underlying network protocols, allowing applications to send and receive network packets at a low level. They are commonly used for network monitoring, packet sniffing, and network security applications.

   Sequenced Packet Sockets: Sequenced packet sockets provide a reliable, connection-oriented communication channel between two endpoints, similar to stream sockets. However, they provide additional features such as message boundaries and record marking, making them

suitable for applications that require message-oriented communication, such as remote procedure calls (RPCs) and interprocess communication (IPC).

6. Discuss about connection oriented and connectionless services with an analogy.
   In computer networking, connection-oriented and connectionless services refer to different ways of transmitting data between two devices.

   Connection-oriented services establish a dedicated communication path between the two devices before transmitting data. This path is maintained throughout the communication, ensuring that data is transmitted reliably and in the correct order. This approach is similar to making a phone call: you dial the number and wait until the other person answers before starting your conversation. During the call, you have a dedicated connection that remains active until you hang up.

   On the other hand, connectionless services do not establish a dedicated communication path before transmitting data. Instead, each data packet is sent independently, without any guarantee of reliability or order. This approach is similar to sending a letter through the mail: you drop the letter into the mailbox and it is sent to its destination without any direct communication between you and the recipient.

   The choice between connection-oriented and connectionless services depends on the needs of the application. For example, applications that require reliable, ordered transmission of data may use connection-oriented services, while applications that prioritize speed and efficiency over reliability may use connectionless services.

   In summary, connection-oriented services are like making a phone call, where a dedicated communication path is established and maintained throughout the communication. Connectionless services are like sending a letter through the mail, where each packet of data is sent independently without a dedicated communication path.
7. Explain in detail about DFA.
   A Deterministic Finite Automaton (DFA) is a mathematical model that represents a finite state machine. A finite state machine is a computational model used to recognize or generate a sequence of symbols or inputs. A DFA can be used to recognize a regular language.

   A DFA is composed of the following components:

   Set of states: The states of the DFA are represented by nodes, and they represent different stages of processing or interpretation. The DFA always starts in one state, and it transitions to another state based on the input symbol it reads.

   Alphabet: The alphabet is a finite set of symbols that are used as inputs to the DFA. Each input symbol belongs to the alphabet, and the DFA can only process symbols from the alphabet.

   Transition function: The transition function maps a pair of a state and an input symbol to a new state. It determines the next state of the DFA based on the current state and the input symbol.

Start state: The start state is the initial state of the DFA. It is where the DFA begins its processing.

Accepting state: The accepting state is a designated state of the DFA that indicates that the DFA has accepted the input. If the DFA reaches an accepting state after processing an input sequence, it means that the input sequence is recognized by the DFA.

The operation of a DFA is based on a set of rules that govern its behavior. The rules are as follows:

The DFA begins in the start state.

The DFA reads the input symbols one at a time.

For each input symbol, the DFA follows the transition function to determine the next state.

If the DFA reaches an accepting state after processing the input sequence, it means that the input sequence is recognized by the DFA.

If the DFA reaches a non-accepting state after processing the input sequence, it means that the input sequence is not recognized by the DFA.

DFA can be used to recognize regular languages. It is a simple and efficient model for recognizing patterns in input strings. Regular expressions can be transformed into DFAs, which are then used to recognize input strings that match the regular expression. DFA is widely used in the field of computer science and automata theory for various applications, such as lexical analysis, string parsing, and pattern recognition.

8. Write a program using typing module to realize dependent programming paradigm.

```
from typing import TypeVar, Generic

T = TypeVar('T')
U = TypeVar('U')

class Pair(Generic[T, U]):
    def __init__(self, first: T, second: U):
        self.first = first
        self.second = second

    def map_first(self, f: Callable[[T], T]) -> 'Pair[T, U]':
        return Pair(f(self.first), self.second)

    def map_second(self, g: Callable[[U], U]) -> 'Pair[T, U]':
        return Pair(self.first, g(self.second))
```

```
    def __str__(self) -> str:
        return f"({self.first}, {self.second})"

def add_one(x: int) -> int:
    return x + 1

def double(x: int) -> int:
    return x * 2

# Create a Pair object
p = Pair(3, 4.5)

# Map the first element using the add_one function
p1 = p.map_first(add_one)
print(p1)  # Output: (4, 4.5)

# Map the second element using the double function
p2 = p.map_second(double)
print(p2)  # Output: (3, 9.0)
```

Part C

1. Write a program using UDP socket for server and client communication.
   Refer lab exercise
2. Write a program using GUI tkinter python to create a hospital management system.

```
from tkinter import *
from tkinter import ttk

class HospitalManagementSystem:
    def __init__(self, root):
        self.root = root
        self.root.title("Hospital Management System")

        # Create a menu bar
        menubar = Menu(root)
        root.config(menu=menubar)

        # Create a file menu
        filemenu = Menu(menubar, tearoff=0)
        filemenu.add_command(label="New", command=self.new_patient)
        filemenu.add_command(label="Exit", command=root.quit)
        menubar.add_cascade(label="File", menu=filemenu)

        # Create a patient form
```

```
        patient_frame = Frame(root, padx=10, pady=10)
        patient_frame.pack()

        # Add patient form fields
        Label(patient_frame, text="Patient Name").grid(row=0, column=0, sticky=W)
        self.patient_name = Entry(patient_frame)
        self.patient_name.grid(row=0, column=1)

        Label(patient_frame, text="Age").grid(row=1, column=0, sticky=W)
        self.patient_age = Entry(patient_frame)
        self.patient_age.grid(row=1, column=1)

        Label(patient_frame, text="Gender").grid(row=2, column=0, sticky=W)
        self.gender = StringVar()
        self.gender.set("Male")
        self.gender_menu = OptionMenu(patient_frame, self.gender, "Male", "Female",
"Other")
        self.gender_menu.grid(row=2, column=1)

        Label(patient_frame, text="Contact Number").grid(row=3, column=0, sticky=W)
        self.contact_number = Entry(patient_frame)
        self.contact_number.grid(row=3, column=1)

        Label(patient_frame, text="Address").grid(row=4, column=0, sticky=W)
        self.address = Text(patient_frame, height=4, width=20)
        self.address.grid(row=4, column=1)

        # Add a submit button
        submit_button = ttk.Button(patient_frame, text="Submit",
command=self.submit_patient)
        submit_button.grid(row=5, column=1)

    def new_patient(self):
        # Clear the patient form fields
        self.patient_name.delete(0, END)
        self.patient_age.delete(0, END)
        self.gender.set("Male")
        self.contact_number.delete(0, END)
        self.address.delete(1.0, END)

    def submit_patient(self):
        # Get the form field values and store them in a database or file
        patient_name = self.patient_name.get()
```

```
        patient_age = self.patient_age.get()
        gender = self.gender.get()
        contact_number = self.contact_number.get()
        address = self.address.get("1.0", "end-1c")

        # Display a success message
        messagebox.showinfo("Success", "Patient record added successfully.")

root = Tk()
app = HospitalManagementSystem(root)
root.mainloop()
```

3. Write about dependent programming paradigm. Write in program to check every key: value pair in a dictionary and check if they match the name : email format.
   The dependent programming paradigm is a programming style in which the types of data and the computations that manipulate them are interdependent. In dependent programming, types are used to encode logical propositions about the data, and the type checker enforces correctness by verifying that these propositions hold.

   Dependent programming languages are typically designed to allow for precise specification of program behavior and to enable type-level computations that can be used to generate code, perform verification, and automate other tasks.

   The dependent programming paradigm is particularly useful in the development of safety-critical systems, where correctness is essential and errors can have catastrophic consequences. It is also gaining popularity in fields such as computer-assisted mathematics, where precise reasoning about types and computations is important.
   For program
   Refer lab exercise
4. Find derivative, integration, limit and solve a quadratic equation using Sympy. Write a program to expand and factorize the following expression. x3 + 3x2y + 3xy2 + y3 = (x + y)3

```
from sympy import *
# make a symbol
x = Symbol('x')
# ake the derivative of sin(x)*e ^ x
ans1 = diff(sin(x)*exp(x), x)
print("derivative of sin(x)*e ^ x : ", ans1)
# Compute (e ^ x * sin(x)+ e ^ x * cos(x))dx
ans2 = integrate(exp(x)*sin(x) + exp(x)*cos(x), x)
print("indefinite integration is : ", ans2)
# Compute definite integral of sin(x ^ 2)dx
# in b / w interval of ? and ?? .
ans3 = integrate(sin(x**2), (x, -oo, oo))
```

```
print("definite integration is : ", ans3)
# Find the limit of sin(x) / x given x tends to 0
ans4 = limit(sin(x)/x, x, 0)
print("limit is : ", ans4)
# Solve quadratic equation like, example : x ^ 2?2 = 0
ans5 = solve(x**2 - 2, x)
print("roots are : ", ans5)
```

Output :

```
derivative of sin(x)*e^x :  exp(x)*sin(x) + exp(x)*cos(x)
indefinite integration is :  exp(x)*sin(x)
definite integration is :  sqrt(2)*sqrt(pi)/2
limit is :  1
roots are :  [-sqrt(2), sqrt(2)]
```

```
def expand_and_factorize(expr):
    # Expand the expression using the binomial theorem
    expanded_expr = "(x + y)^3 = " + "x^3 + 3x^2y + 3xy^2 + y^3"

    # Simplify the expanded expression
    simplified_expr = "(x + y)^3 = " + expr
    if expr == "x^3 + 3x^2y + 3xy^2 + y^3":
        simplified_expr += " = (x + y)^3"
    elif expr == "(x + y)^3":
        simplified_expr += " = x^3 + 3x^2y + 3xy^2 + y^3"
    else:
        simplified_expr += " is not a valid expression."

    # Return the expanded and simplified expression
    return expanded_expr + "\n" + simplified_expr
expr = "x^3 + 3x^2y + 3xy^2 + y^3"
result = expand_and_factorize(expr)
print(result)
```