1.SymPy is a collection of mathematical algorithms and convenience functions built on the ---------- extension of Python

a)numpy b)scikit
c)sys d) functools

2. Exponential function computes the ----------

a) 10x element-wise** b) 10**x row-wise
c) 10**x column-wise d) 10*x element-wise

3._____ evaluates the expression to a floating-point number.

a)evalf b)fval

c) float d) valf

4. what is the output for the following expression ((x+y)**2).expand()

a) x,2 + 2,x,y + y,2 b) x*2 + 2*x*y + y*2

c) x^2 + 2*x*y + y^2 **d) x**2 + 2*x*y + y**2**

5. limit((5**x + 3**x)**(1/x), x, oo) ,what is the output

a)5 b) oo

c) 1 d) 0

6) Higher derivatives can be calculated using the which method

a) highder(func,var,n) **b) diff(func, var, n)**

c) diff(n,var,func) d) diff(func, var)

7) what is the output

```
>>> x = Symbol('x')
>>> y = Symbol('y')
>>> A = Matrix([[1,x], [y,1]])
>>> A**2
```

a) [1, x]
[y, 1]

b) [xy + 1, 2x]
[2y, xy + 1]

c) [x*y + 1, 2*y]

d) [x*y + 1, 2*x]

[2*x, x*y + 1]

[2*y, x*y + 1]

8) .match() method, along with the ------- class, to perform pattern matching on expressions.

a) pattern

b) func

c) wild

d) dictionary

9) which among the following function Return or print, respectively, a pretty representation of expr

a) pretty(expr)

b) pretty_print(expr)

c) pprint(expr)

d) all of the above

10) What is the output of

from sympy.abc import a, b

expr = b*a + -4*a + b + a*b + 4*a + (a + b)*3

a) ba-4a+b+ab+4a+3(a+b)

b) 2*a*b + 3*a + 4*b

c) 2ab+3a+4b

d) all of the above

11) print(pi.evalf(30))

a) 3.14/30

b)30/3.14

c) 3.14159265358979323846264338328

d) 3.14

12) which is the correct way to write equation for x^2=x in sympy

a) x2 =x**

b) x*x = x

c) x%2 = x

d) X^2 =x

13) how to find a solution for a equation in a given interval

a) solve(equation,range)

b)equation(solve,range)

c) solveset()

d) both a and b

14) Allows , the same elements can appear multiple times at different positions

a) set

b)sequence

c) dictionary

d) none

15) which is the snippet to find the eigenvalues of $\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$

a) `Matrix([[1, 2], [2, 2]]).eigenvals()`

b) `Matrix([[1, 2], [2, 2]]).eigen`

c) both a and b

d) eigen(([[1, 2], [2, 2]])

16 What is the purpose of sympify() method?

a. **Convert expression of string type to mathematical expression**
b. Convert mathematical expression to String
c. Convert mathematical expression to character
d. Convert tuple to mathematical expression

17 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}]

18. A *rational expression* is an algebraic expression in which the numerator and denominator are both ----------

a. Equal
b. **Polynomials**
c. Unequal
d. Symmetric

19. Find the output of the following program

```
# import sympy
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

20. ---------- method will simplify mathematical expression using trigonometric identities.

a. sympy.trigsimp()
b. sympy.series()
c. sympy.lambda()
d. sympy.sim()

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

2) Which of the following is false regarding dependent types?

a) They allow us to write programs and know they are correct before running them.
b) **They allow us to write programs and know they are correct only after running them**.
c) You can specify types that can check the value of your variables at compile time.
d) Its definition depends on a value.

3) Which of the notations is true for "P(x) is true for all values of x in the universe of discourse"?

a) x∀P(x)
b) ∀P(x)x
c) ∃xP(x)
d) ∀xP(x)

4) Which of the following is false about quantifiers?

a) Notation ∀ is used for Universal quantifier.
b) Notation ∃is used for Existential quantifier.
c) "All men drink tea" is an example of Universal Quantifier
d) "All men drink coffee" is an example of Existential Quantifier.

5) Let P(x) be the predicate "x must take a discrete mathematics course" and let Q(x) be the predicate "x is a computer science student".
Which of the following statements is correct for "Everybody must take a discrete mathematics course or be a computer science student"?

a) ∀x(Q(x) V P(x))
b) ∀x(Q(x)) V ∀x(P(x))
c) ∀x(Q(x) || P(x))
d) ∀x(Q(x) -> P(x))

6) Which of the following is correct for predicate "All men drink coffee"?

a) ∀x men(x) → (x, coffee)
b) drink (x, coffee)→∀x men(x)

c) ∀x men(x) → drink (x, coffee)
d) ∀men(x) → drink (x, coffee)

7) 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)

8) Dependent Type is used to encode _____ like "for all" and "there exists".

a. Logic Quantifiers

b. Analysing Quantifiers

c. Dependent Quantifiers

d. None of the above

9) “There exists x in the universe of discourse such that P(X) is true” is which Quantifiers statement?

a. Logical

b. Universal

c. Existential

d. None of these

10) What is the way to determine if a given function is dependant type

a. Result is independent of the argument
b. Result depends upon the usage in the program
c. Result depends on the Value of its argument
d. Result depends on available resources

11) Notation for an Existential Quantifier:

a.∀xP(x)
b.ΣP(x)
c.ØxP(x)
d.∃xP(x)

12) Representation of the following statement:
Every Clock has quartz

a. Øxclock(x)->quartz(x)
b. ∃xclock(x)->quartz(x)
c. ∀xclock(x)->quartz(x)
d. none of the above

13) Representation of the following statement:
Some leaves are Red

a. ∃xleaves(x)->red(x)
b. ∀xleaves(x)->red(x)
c. Σleaves(x)->red(x)
d. none of the above

13. A function has dependent type if the _____of a function's result depends on the _____ of its Argument

a. value and type

b. type and value

c. type and type

d. value and value

14. Dependent type paradigm used to encode logic's quantifiers like _____and _____

a. for one, there may exists

b. for all, there always

c. for all, there exists

d. for specific, there exists

15. Choose the correct one with respect to typing and typing-extensions library in python dependent type programming.

a. typing is not builtin python module where all possible types are defined.
typing_extensions is an official package for new types in the future releases of python
b. typing is a builtin python module where all possible types are defined.
typing_extensions is an official package for new types in the future releases of python
c. typing is a builtin python module where all possible types are defined.
typing_extensions is not an official package for new types in the future releases of python
d. typing is not a builtin python module where all possible types are defined.
typing_extensions is not an official package for new types in the future releases of python

16. Predict the output of the below mentioned python code without dependent type syntax:

```
def make_hamburger(meat, number_of_meats):
   return ["bread"] + [meat] * number_of_meats + ["bread"]
print(make_hamburger("ground beef", 2))
```

```
a. ['bread', 'ground beef', 2, 'bread']
b. ['bread', 'ground beef', 'ground beef', 'bread']
c. TypeError: cannot concatenate 'str' and 'int' objects
```

```
d. ['bread', 'ground beef', '2bread']
```

17. Predict the output of the below mentioned python code with dependent type syntax:

```
from typing import List

def greet_all(names: List[str]) -> None:
    for name in names:
        print('Hello ' + name)

names = ["Alice", "Bob", "Charlie"]
ages = [10, 20, 30]

greet_all(names)
greet_all(ages)


a.greet_all(names)   # Ok!
  greet_all(ages)   # Ok!

b. a.greet_all(names)   # Ok!
  greet_all(ages)   # Error due to incompatible types

c. a.greet_all(names)  # Error due to incompatible types
  greet_all(ages)   # Ok!

d. a.greet_all(names)   # Error due to incompatible types
  greet_all(ages)   # Error due to incompatible types
```

18 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).

a) ∀x → work (x, industry) d)∀x man(x) → Industry(work)

19 How do you represent the statement "Every man respects his parent."

a) ∀x woman(x) → respects (x, parent) **b) ∀x man(x) → respects (x, parent)**

c) ∀x man → respects (x, parent) d) ∀x man(x) → respects (parent).

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)

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

VADAPALANI CAMPUS

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

18CSC207J – ADVANCED PROGRAMMING PRACTICES

NETWORK PROGRAMMING PARADIGM

1. _____ programming is a way of connecting two nodes on a network to communicate with each other.
    i. Logic
    ii. **Socket**
    iii. Functional
    iv. Symbolic

2. Which method used by the server to initiate the connection with the client?
    i. Listen()
    ii. Close()
    iii. Bind()
    iv. **Accept()**

3. _____is a socket through which data can be transmitted continuously.
    i. Datagram Socket
    ii. Stream Socket
    iii. Raw Socket
    iv. Binary Socket

4. _____ paradigm deals with client – server communication
    i. Dependent type paradigm
    ii. Parallel programming paradigm
    iii. Network paradigm
    iv. Concurrent programming paradigm

5. _____ protocol facilitates sending of datagrams in an unreliable manner.

i. TCP
ii. UDP
iii. HTTP
iv. FTP

6. Which among methods are not server socket?
i. connect( )
ii. bind( )
iii. listen()
iv. accept()

7. In UDP, Which among methods are used to receive messages at endpoint
i. sock_object.recv()
ii. sock_object.send()
iii. **sock_object.recvfrom()**
iv. sock_object.sendto()

8. In TCP, Which among methods are used to send messages from endpoint
i. sock_object.recv()
ii. sock_object.send()
iii. sock_object.recvfrom()
iv. sock_object.sendto()

9. The protocol which defines IPv4 is
i. AF_UNIX
ii. SOCK_STREAM
iii. AF_INET
iv. SOCK_DGRAM

10. The correct order of methods used in server socket is
i. Socket(), Bind(), listen(), accept()
ii. Socket(), listen(), bind(), accept()
iii. Socket(), accept(), bind(), listen()
iv. Socket(), bind(), accept(), listen()

11. _____ is a type of network socket which provides connection less point for sending and receiving packets.
    i. **Datagram Socket**
    ii. Stream Socket
    iii. Raw Socket
    iv. Binary Socket
12. _____ do not use any transport protocol but data is directly transmitted over IP protocol
    i. Datagram Socket
    ii. Stream Socket
    iii. Raw Socket
    iv. Binary Socket
13. The _____ is a physical path over which the message travels.
    i. Protocol
    ii. **Medium**
    iii. Path
    iv. Route
14. A pair (host, port) is used for the _____ address family.
    i. AF_NETLINK
    ii. AF_INET6
    iii. **AF_INET**
    iv. AF_ALG
15. Use _____ to make the socket to visible to the outside world.
    i. socket.listen()
    ii. socket.visible()
    iii. socket.socket()
    iv. **Socket.gethostname()**
16. In which mode socket is created in default.
    i. **Blocking mode**
    ii. Non-Blocking Mode
    iii. Timeout mode
    iv. Accept mode
17. Which method is recommended to be called before calling connect() method?
    i. getdefaulttimeout()

ii. **settimeout()**

iii. getaddrinfo()

iv. no such method

18. Which exception is raised for address related errors by getaddrinfo()?

i. gaoerror

ii. gsierror

iii. **gaierror**

iv. gdeerror

19. _____ protocol facilitates sending of datagrams in an reliable manner.

i. TCP

ii. UDP

iii. HTTP

iv. FTP

20. To create a socket, which function among the following is available in python socket module?

i. socket.create()

ii. socket.initialize()

iii. **socket.socket()**

iv. socket.build()

Unit 4

- Logic Programming Paradigm
- Dependent Type Programming Paradigm
- Network Programming Paradigm

1. Logic Programming Paradigm

It can be termed as abstract model of computation. It would solve logical problems like puzzles, series etc. In logic programming we have a knowledge base which we know before and along with the question and knowledge base which is given to machine, it produces result.

In normal programming languages, such concept of knowledge base is not available but while using the concept of artificial intelligence, machine learning we have some models like Perception model which is using the same mechanism.
In logical programming the main emphasize is on knowledge base and the problem.

The execution of the program is very much like proof of mathematical statement, e.g., Prolog

Syntax for Logic Programming Paradigm

```
sum of two number in prolog:
  predicates
  sumoftwonumber(integer, integer)
clauses


  sum(0, 0).
   sum(n, r):-
        n1=n-1,
        sum(n1, r1),
        r=r1+n
```

- It can be an abstract model of computation.
- Solve logical problems like puzzles

- Have knowledge base which we know before and along with the question you specify knowledge and how that knowledge is to be applied through a series of rules
- The Logical Paradigm takes a declarative approach to problem-solving.
- Various logical assertions about a situation are made, establishing all known facts.
- Then queries are made.

Logic programming is a paradigm where computation arises from proof search in a logic according to a fixed, predictable strategy. A logic is a language. It has syntax and semantics. It. More than a language, it has inference rules.

- **Syntax:**

  Syntax: the rules about how to form formulas; usually the easy part of a logic.

- **Semantics:**

  About the meaning carried by the formulas, mainly in terms of logical consequences.

- **Inference rules:**

  Inference rules describe correct ways to derive conclusions.

Lucy is a Professor
Danny is a Professor

James is a Lecturer
All professors are Dean
Write a Query to reterive all deans?

Solution

```
from pyDatalog import pyDatalog
pyDatalog.create_terms('X,Y,Z,professor,lecturer, dean')
+professor('lucy')
+professor('danny')
+lecturer('james')
dean(X)<=professor(X)
print(dean(X))
likes(john, susie).
likes(X, susie).
likes(john, Y).
likes(john, Y), likes(Y, john).
likes(john, susie); likes(john,mary).
not(likes(john,pizza)).
likes(john,susie) :- likes(john,mary). likes Mary.
rules
friends(X,Y) :- likes(X,Y),likes(Y,X).
hates(X,Y) :- not(likes(X,Y)).
enemies(X,Y) :- not(likes(X,Y)),not(likes(Y,X)).
```

ALGORITHM 1:

- Import pyDatalog and pyDatalog
- Creating new pyDatalog variables
- Creating square function
- Reverse order as desired
- Displaying the output.
- Print the output.

PROGRAM 1:

```
from pyDatalog import pyDatalog
pyDatalog.create_terms("A, Pow2, Pow3")
```

```
def square(n):
    return n*n
(predefined logic) as pyDatalogvarible {Optional}
pyDatalog.create_terms("square")
input_values = range(10)[::-1]
 (querying with & operator)
print ( A.in_(input_values) & (Pow2 == square(A)) & (Pow3 == A**3) ).
```

2.Dependent Type Programming Paradigm

In computer science and logic, a dependent type is a type whose definition depends on a value.

It is an overlapping feature of type theory and type systems. Used to encode logic's quantifiers like "for all" and "there exists".

Dependent types may help reduce bugs by enabling the programmer to assign types that further restrain the set of possible implementations.

Quantifiers

A predicate becomes a proposition when we assign it fixed values. However, another way to make a predicate into a proposition is to quantify it. That is, the predicate is true (or false) for all possible values in the universe of discourse or for some value(s) in the universe of discourse. Such quantification can be done with two quantifiers: the universal quantifier and the existential quantifier.

Universal: Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing. The universal quantification of a predicate P(x) is the proposition "P(x) is true for all values of x in the universe of discourse" We use the notation ∀ for Universal quantifier

$$\forall x P(x)$$

which can be read "for all x"

Example:

Let P(x) be the predicate "x must take a discrete mathematics course" and let Q(x) be the predicate "x is a computer science student". The universe of discourse for both P(x) and Q(x) is all UNL students.

Express the statement "Every computer science student must take a discrete mathematics course".

$$\forall x(Q(x) \rightarrow P(x))$$

Express the statement "Everybody must take a discrete mathematics course or be a computer science student".

$$\forall x(Q(x) \vee P(x))$$

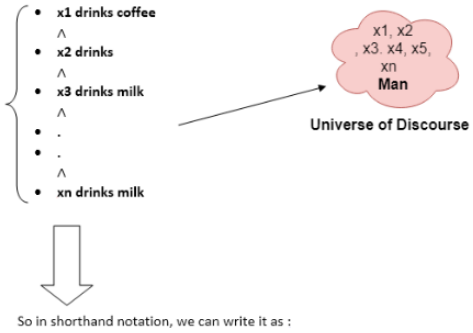
If x is a variable, then ∀x is read as:

For all x **For each x** **For every x.**

Quantifiers

All man drink coffee.

Let a variable x which refers to a cat so all x can be represented in UOD as below:



∀x man(x) → drink (x, coffee).
It will be read as: There are all x where x is a man who drink coffee.

Examples

1. All birds fly.

In this question the predicate is "fly(bird)."

And since there are all birds who fly so it will be represented as follows.

∀x bird(x) →fly(x).

2. Every man respects his parent.

In this question, the predicate is "respect(x, y)," where x=man, and y= parent.

Since there is every man so will use ∀, and it will be represented as follows:

∀x man(x) → respects (x, parent).

3. Some boys play cricket.

In this question, the predicate is "play(x, y)," where x= boys, and y= game. Since there are some boys so we will use ∃, and it will be represented as:

∃x boys(x) → play(x, cricket).

Write the python program to Program to create type aliases typing module.

ALGORITHM 1:

- Start
- From typing import list
- Initiate vector e
- def scale function initialize to vector :
  Return [scalar* number of or number in vector]
- Declare a scale function
- Print output
- End the program

PROGRAM 1:

```
from typing import List
# Vector is a list of float values
Vector = List[float]
def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]
a = scale(scalar=2.0, vector=[1.0, 2.0, 3.0])
print(a)
```

OUTPUT 1 :

[2.0, 4.0, 6.0]

2.Program to check every key: value pair in a dictionary and check if they match the name : email format.

AIM:

To write a program to check every key: value pair in a dictionary and check if they match the name : email format.

ALGORITHM 2:

- Start
- From typing import disk
- import rc
- create an alias called ‘Contact Diet’
- check if name and strings
- check for email
- Print the output

PROGRAM 2:

```
from typing import Dict
import re
 # Create an alias called 'ContactDict'
ContactDict = Dict[str, str]
 def check_if_valid(contacts: ContactDict) -> bool:
    for name, email in contacts.items():
        # Check if name and email are strings
        if (not isinstance(name, str)) or (not isinstance(email, str)):
            return False
        # Check for email xxx@yyy.zzz
        if not re.match(r"[a-zA-Z0-9\._\+-]+@[a-zA-Z0-9\._-]+\.[a-zA-Z]+$", email):
            return False
    return True
print(check_if_valid({'viji': 'viji@sample.com'}))
print(check_if_valid({'viji': 'viji@sample.com', 123: 'wrong@name.com'}))
```

OUTPUT 2:

True
False

Network Programming Paradigm :

The Network paradigm involves thinking of computing in terms of a client, who is essentially in need of some type of information, and a server, who has lots of information and is just waiting to hand it out. Typically, a client will connect to a server and query for certain information. The server will go off and find the information and then return it to the client.

In the context of the Internet, clients are typically run on desktop or laptop computers attached to the Internet looking for information, whereas servers are typically run on larger computers with certain types of information available for the clients to retrieve. The Web itself is made up of a bunch of computers that act as Web servers; they have vast amounts of HTML pages and related data available for people to retrieve and browse. Web clients are used by those of us who connect to the Web servers and browse through the Web pages.

Network programming uses a particular type of network communication known as sockets. A socket is a software abstraction for an input or output medium of communication.

Socket

- A socket is a software abstraction for an input or output medium of communication.
- Sockets allow communication between processes that lie on the same machine, or on different machines working in diverse environment and even across different continents.
- A socket is the most vital and fundamental entity. Sockets are the end-point of a two-way communication link.
- An endpoint is a combination of IP address and the port number.

For Client-Server communication,

- Sockets are to be configured at the two ends to initiate a connection,
- Listen for incoming messages
- Send the responses at both ends
- Establishing a bidirectional communication.

Socket Types

Datagram Socket

- A datagram is an independent, self-contained piece of information sent over a network whose arrival, arrival time, and content are not guaranteed. A datagram socket uses User Datagram Protocol (UDP) to facilitate the sending of datagrams (self-contained pieces of information) in an unreliable manner. Unreliable means that information sent via datagrams isn't guaranteed to make it to its destination.

Stream Socket:

- A stream socket, or connected socket, is a socket through which data can be transmitted continuously. A stream socket is more akin to a live network, in which the communication link is continuously active. A stream socket is a "connected" socket through which data is transferred continuously.

sock_obj =socket.socket(socket_family, socket_type, protocol=0)

socket_family: - Defines family of protocols used as transport mechanism.

Either AF_UNIX, or
AF_INET (IP version 4 or IPv4).

socket_type: Defines the types of communication between the two end-points.

SOCK_STREAM (for connection-oriented protocols, e.g., TCP), or

SOCK_DGRAM (for connectionless protocols e.g. UDP).

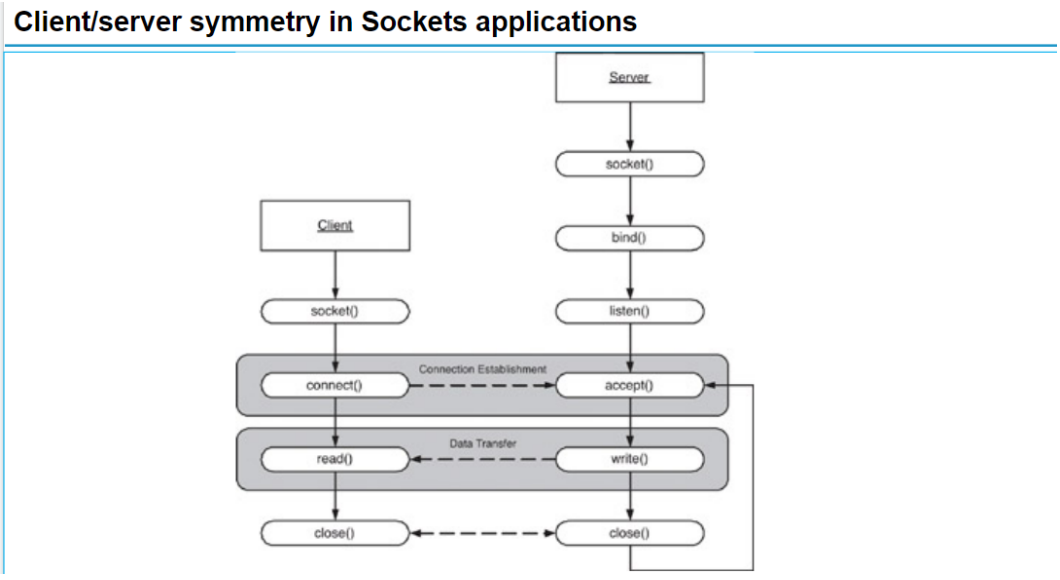
protocol: We typically leave this field or set this field to zero.

Example:

```
#Socket client example in python
import socket
#create an AF_INET, STREAM socket (TCP)
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print 'Socket Created'.
```

```
import socket
import sys
try:
    #create an AF_INET, STREAM socket (TCP)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
except socket.error, msg:
    print 'Failed to create socket. Error code: ' + str(msg[0]) + ' , Error message : ' + msg[1]
    sys.exit();

print 'Socket Created'
```



sock_object.recv():

Use this method to receive messages at endpoints when the value of the protocol parameter is TCP.

sock_object.send():

Apply this method to send messages from endpoints in case the protocol is TCP.

sock_object.recvfrom():

Call this method to receive messages at endpoints if the protocol used is UDP.

sock_object.sendto():

Invoke this method to send messages from endpoints if the protocol parameter is UDP.

sock_object.gethostname():

This method returns hostname.

sock_object.close():

This method is used to close the socket. The remote endpoint will not receive data from this side.

IMPLEMENTATION OF NETWORK PROGRAMMING PARADIGM

AIM:

To implement network programming paradigm in python.

1. Program to implement client server program using TCP

ALGORITHM 1:

TCP Server:

- Import Socket
- Create a socket object
- Reserve a port on your computer
- Next bind to the port this makes the server listen to requests coming from other computers on the network
- Put the socket into listening mode
- Establish connection with client.
- Send a thank you message to the client.
- Close the connection with the client
- Print the output.

TCP Client:

- Import Socket
- Create a socket object
- Define the port on which you want to connect
- Connect to the server on local computer
- Receive data from the server
- Close the connection with the server
- Print the output

PROGRAM 1:

TCP server

```
import socket

s = socket.socket()

print("Socket successfully created")

port = 12345

s.bind(('', port))

print("socket binded to %s" %(port))

s.listen(5)

print("socket is listening")

while True:

   c, addr = s.accept()

print("Got connection from", addr)

c.send('Thank you for connecting')

c.close()
```

OUTPUT :

```
Socket successfully created
socket binded to 12345
socket is listening
Got connection from ('127.0.0.1', 52617)
```

TCP Client

```
# Import socket module
import socket
# Create a socket object
s = socket.socket()

# Define the port on which you want to connect

port = 12345
```

```
# connect to the server on local computer

s.connect(('127.0.0.1', port))

# receive data from the server

Print( s.recv(1024) )

# close the connection

s.close()
```

OUTPUT:

Thank you for connecting

Program to implement client server program using UDP

2. Program to implement client server program using UDP

UDP Server:

- Import Socket
- Create a datagram socket
- Bind to address and ip address
- Listen for incoming datagram
- Send reply to client
- Print client message
- Print the client IP address.

UDP Client:

- Import Socket
- Create a client socket
- Create a UDP socket at client side
- Send to server using created UDP socket
- Print the server message

UDP Server:

```
import socket
```

```
localIP     = "127.0.0.1"

localPort   = 20001

bufferSize  = 1024

msgFromServer       = "Hello UDP Client"

bytesToSend         = str.encode(msgFromServer)

UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

UDPServerSocket.bind((localIP, localPort))

Print("UDP server up and listening")

while(True):

bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)

message = bytesAddressPair[0]

address = bytesAddressPair[1]

clientMsg = "Message from Client:{}".format(message)

clientIP  = "Client IP Address:{}".format(address)

print(clientMsg)

print(clientIP)

UDPServerSocket.sendto(bytesToSend, address)
```

OUTPUT:

```
UDP server up and listening

Message from Client:b"Hello UDP Server"

Client IP Address:("127.0.0.1", 51696)
```

UDP Client

```
import socket

msgFromClient       = "Hello UDP Server"
```

```
bytesToSend         = str.encode(msgFromClient)

serverAddressPort   = ("127.0.0.1", 20001)

bufferSize          = 1024

UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

UDPClientSocket.sendto(bytesToSend, serverAddressPort)

msgFromServer = UDPClientSocket.recvfrom(bufferSize)

msg = "Message from Server {}".format(msgFromServer[0])

print(msg)
```

OUTPUT:

```
Message from Server b"Hello UDP Client"
```

Unit 5

- Symbolic Programming paradigm
- Automata Programming Paradigm
- GUI Programming paradigm

Symbolic Programming Paradigm

Symbolic programming is **a programming paradigm in which the program can manipulate its own formulas and program components as if they were plain data.**

Symbolic programming **uses linguistic structures as the foundation of all aspects of computation**. From a computation's description, to how the computation executes, to how humans interface with the results, the exact same basic tree structure is used throughout.

Symbolic computation deals with the computation of mathematical objects symbolically. This means that the mathematical objects are represented exactly, not approximately, and mathematical expressions with unevaluated variables are left in symbolic form.

It Covers the following:

- As A calculator and symbols
- Algebraic Manipulations - Expand and Simplify
- Calculus – Limits, Differentiation, Series , Integration
- Equation Solving – Matrices

Rational - ½, or 5/2

```
>>import sympy as sym
>>a = sym.Rational(1, 2)
>>a                                  Answer will be 1/2
```

LIMITS compute the limit of

limit(function, variable, point)

limit(sin(x)/x , x, 0) =1

Differentiation

diff(func,var) eg diff(sin(x),x)=cos(x)

diff(func,var,n) eg

Series

series(expr,var)

series(cos(x),x) = 1-x/2+x/24+o(x)

Integration

Integrate(expr,var)

Integrate(sin(x),x) = -cos(x)

ALGORITHM:

- Import sympy module
- Evaluate the expression using sympy command
- Print the result

PROGRAM:

```
from sympy import *
expr = cos(x) + 1
print( expr.subs(x, y))
x, y, z = symbols("x y z")
 print(expr = x**y)
print(expr)
expr = sin(2*x) + cos(2*x)
print( expand_trig(expr))
print(expr.subs(sin(2*x), 2*sin(x)*cos(x)))
expr = x**4 - 4*x**3 + 4*x**2 - 2*x + 3
 replacements = [(x**i, y**i) for i in range(5) if i % 2 == 0]
print( expr.subs(replacements))
str_expr = "x**2 + 3*x - 1/2"
 expr = sympify(str_expr)
```

```
expr
 expr.subs(x, 2)
expr = sqrt(8)
print( expr.evalf())
expr = cos(2*x)
 expr.evalf(subs={x: 2.4})
one = cos(1)**2 + sin(1)**2
print( (one - 1).evalf())
```

OUTPUT:

cos(y)+1

Xxy

2sin(x)cos(x)+2cos2(x)−1

2sin(x)cos(x)+cos(2x)

4x3−2x+y4+4y2+3

192

2.82842712474619

0.0874989834394464

x2+3x−12

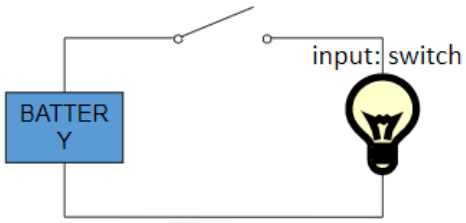
Automata Programming Paradigm

Automata-based programming is a programming paradigm in which the program or its part is thought of as a model of a finite state machine or any other formal automation.

What is Automata Theory?

- Automata theory is the study of abstract computational devices
- Abstract devices are (simplified) models of real computations . Computations happen everywhere: On your laptop, on your cell phone, in nature

Example:



output: light bulb

actions: flip switch

states: on, off

Alphabets and strings

A common way to talk about words, number, pairs of words, etc. is by representing them as strings

To define strings, we start with an alphabet

An alphabet is a finite set of symbols.

Examples:

$\Sigma_1 = \{a, b, c, d, \ldots, z\}$: the set of letters in English
$\Sigma_2 = \{0, 1, \ldots, 9\}$: the set of (base 10) digits
$\Sigma_3 = \{a, b, \ldots, z, \#\}$: the set of letters plus the special symbol #
$\Sigma_4 = \{ (,) \}$: the set of open and closed brackets

Strings

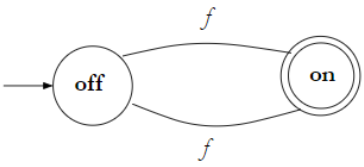
A string over alphabet Σ is a finite sequence of symbols in Σ.

The empty string will be denoted by e

Examples:

abfbz is a string over $\Sigma_1 = \{a, b, c, d, \ldots, z\}$
9021 is a string over $\Sigma_2 = \{0, 1, \ldots, 9\}$
ab#bc is a string over $\Sigma_3 = \{a, b, \ldots, z, \#\}$
))()(() is a string over $\Sigma_4 = \{ (,) \}$

Finite Automata



There are states off and on, the automaton starts in off and tries to reach the “good state” on

What sequences of fs lead to the good state?

Answer: {f, fff, fffff, ...} = {f n: n is odd}

This is an example of a deterministic finite automaton over alphabet {f}

- A deterministic finite automaton (DFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where
  - Q is a finite set of states
  - Σ is an alphabet
  - $\delta: Q \times \Sigma \rightarrow Q$ is a transition function
  - $q_0 \in Q$ is the initial state
  - $F \subseteq Q$ is a set of accepting states (or final states).
- In diagrams, the accepting states will be denoted by double loops

Example of DFA using Python

```
from automata.fa.dfa import DFA
# DFA which matches all binary strings ending in an odd number of '1's
dfa = DFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'0', '1'},
    transitions={
        'q0': {'0': 'q0', '1': 'q1'},
        'q1': {'0': 'q0', '1': 'q2'},
        'q2': {'0': 'q2', '1': 'q1'}
    },
    initial_state='q0',
    final_states={'q1'}
)
dfa.read_input('01')   # answer is  'q1'
dfa.read_input('011')  # answer is error
print(dfa.read_input_stepwise('011'))
Answer # yields:
# 'q0'     # 'q0'     # 'q1'
# 'q2'     # 'q1'
```

```
if dfa.accepts_input('011'):
    print('accepted')
else:
    print('rejected')
```

DFA refers to **deterministic finite automata**. · In **DFA**, there is only one path for specific input from the current state to the next state.

These machines are called finite because there are a limited number of possible states which can be reached. A finite automaton is only called deterministic if it can fulfill both conditions. DFAs differ from non-deterministic automata in that the latter are able to transition to more than one state at a time and be active in multiple states at once.

In practice, DFAs are made up of five components (and they're often denoted by a five-symbol set known as a 5-tuple). These components include:

- A finite number of states
- A set of symbols known as the alphabet, also finite in number
- A function that operates the transition between states for each symbol
- An initial start state where the first input is given or processed
- A final state or states, known as accepting states.

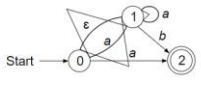
NDFA

- A nondeterministic finite automaton M is a five-tuple M = (Q, Σ, δ, q_0, F), where:
  - Q is a finite set of states of M
  - Σ is the finite input alphabet of M
  - δ: Q × Σ → power set of Q, is the state transition function mapping a state-symbol pair to a subset of Q
  - q_0 is the start state of M
  - F ⊆ Q is the set of accepting states or final states of M

NDFA

A nondeterministic finite automaton (NFA) over an alphabet A is similar to a DFA except that epislon-edges are allowed, there is no requirement to emit edges from a state, and multiple edges with the same letter can be emitted from a state.

Example. The following NFA recognizes the language of a + aa*b + a*b.



	T	a	b	ε
start	0	{1, 2}	∅	{1}
	1	{1}	{2}	∅
final	2	∅	∅	∅

Table representation of NFA

An NFA over A can be represented by a function T : States × A ∪ {L} → power(States), where T(i, a) is the set of states reached from state i along the edge labeled a, and we mark the start and final states. The following figure shows the table for the preceding NFA.

Example of NFA using Python

```
from automata.fa.nfa import NFA
# NFA which matches strings beginning with 'a', ending with 'a', and containing
# no consecutive 'b's
nfa = NFA(
    states={'q0', 'q1', 'q2'},
    input_symbols={'a', 'b'},
    transitions={
        'q0': {'a': {'q1'}},
            # Use '' as the key name for empty string (lambda/epsilon) transitions
        'q1': {'a': {'q1'}, '': {'q2'}},
        'q2': {'b': {'q0'}}
    },
    initial_state='q0',
    final_states={'q1'}
)
```

```
nfa.read_input('aba')
ANSWER :{'q1', 'q2'}

nfa.read_input('abba')
ANSWER: ERROR


nfa.read_input_stepwise('aba')

if nfa.accepts_input('aba'):
    print('accepted')
else:
    print('rejected')
ANSWER: ACCEPTED
nfa.validate()
ANSWR: TRUE
```