Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment #1: Implementation of Basic Python Programs

Aim/Objective:

Implementation of Basic Python Programs.

Description:

Students will learn and understand the basic concepts in Python programming language.

Pre-Requisites:

- Basic Computer Skills
- Basic Mathematics
- Logical Thinking
- Text Editor or Integrated Development Environment (IDE)

Pre-Lab:

1. What is the purpose of implementing basic python programs in this lab?

A) The purpose of Implementing basic python Programs in th lab is to practise the funda -mental Concepts such as data structure Algorithms and Control structures in a pythonic way.

2. What is python, and how do you comment in Python?

A) Python is a high-level interpreted language that is easy to learn and understand know for it's Simplicity readability and large Community support.

3. What are variables in Python? How do you print output in Python?

In Python variables are names given to objects and you can print output using the print() function, for Ex. x=5; print(x) would prints.

4. How do you take user input in Python? What are data types in Python?

In python we make the user input function Python has several built-in data types int float (str, bool) list tuple.

5. How do you create a function in Python? How do you define a conditional statement in Python?

In python we can define a conditional statement using if elif and else and also using the def keyword.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In-Lab:

Implementation of Basic Python Programs.

1. Write a python Program to Find Largest of 3 Numbers using nested if-else
2. Write a python Program to Swap Two Variables using Third Variable
3. Write a python Program to find FACTORIAL of a given number.
4. Write a python Program to find the PRIME NUMBERS in the given range
5. Write a python Program to Print Fibonacci Series up to N Term
6. Create a list of integers and *append, insert*, and *remove* elements from the list. Acce elements using *indexing* and *slicing* .
7. Perform the list methods of *append(), extend(), insert(), remove(), pop(), clear(), in count(), sort(), reverse(), copy()* on a sample list and observe the changes.
8. Create a tuple of integers and Access elements using indexing and slicing.
9. Create a dictionary with tuples as keys and access and modify the dictionary using t keys.
10. Define a function that takes your name as input and returns a greeting message

Procedure/Program:

(1) Algorithm

1. The Algorithm Compares first a and b
2. The a is greater than or Equal to b
3. Then it Compares 'c'.
4. It Compares and it Returns the largest Number.

code:

```
def largest_of_three(a,b,c)
    if a>=b:
        if a>=c
            return a
    elif:
        return c
    else:
        return c
```

Output:

10, 20, 30

= 30

② Algorithm

1. Take the Numbers
2. Give the third variable
3. Let the Compares all the values
4. Finally Executes the Algorithm

Code:

```
a:5
b=10
Print("Before Swapping")
  Print(a)
  Print(b)
Print("After Swapping")
  temp=a
  a=b
  b= temp
  Print values
```

Output:

```
Before Swapping
   5
   10
After Swapping
   10
   5
```

③ Algorithm

1. The function initilizes a variable 'result' to 1
2. It then loops 1 to 'n' Multiplying 'Result' with Each number in the Range
3. Finaly it Returns the Calculated factorial.

Code:

```
def factorial(n):
  Result = 1
  for i in range(1, n+1):
    Result *= i
  Return Result.
```

Output:

```
5
= 120
```

④ Algorithm

1. The function Return 'false' if the input n is less than 2 Because prime no are Greater than or Equal 2
2. The function loops from 2 to the Square of 'n' Cheking if n is divisible of any of

3. If the function has not returned 'False' by the end of loop it returns 'True' as 'n' is a prime number.

Code:

```
def is_prime(n):
    if n<2:
        return false
for i in range(2, int(n** 0.5)+1):
    if n%.i' =0',
        return false
    return true.
```

Output:

5 = True
6 - false.

④ Algorithm

1. It takes an integer 'n' as input which represents no of terms in Fibonacci to generate.

2. If n is less than or Equal to '0' returns error Message

3. The function then Enters a loop, Appending a Sum of the last two terms in the Sequence to the list the desired length.

Code:

```
def fibonacci
    if n<0:
        return :Input should
    be a positive
Integer.
elif n=1;
    return 0;
elif n=2:
    return 1
else
    fib_Sequence = [0,1]
    while len(fib_Sequence
fib_sequence.append(fib_
    Sequence[-1]+ fib_Sequence
    return fib_Sequence
```

Output:

[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]

⑤ Algorithm

1. Creating a list of Integers
2. Appending using append()
3. insert() Inserting
4. Removes() Removing
5. Indexing and Slicing the Elements and values being the position of list.

Code:

```
def My list = [1,2,3,4,5]
My.list.append(6)
Print My list [1,2,3,4,5,6]
My.list.Insert(2,7)
Print (My-list) [1,2,7,3,4,5,6]
My.list Remove(4)
Print (My-list ] [1,2,7,3,5,6]
Print my list [0] = 1
Print my list [-1] = 6
Print (My-list [1:3]) [2,7]
Print My-list [:3]) [3,5,6]
Print [My-list [3:]) [1,2,7)
Print My list [1,2,3,5,6]
```

Output:

1,2,3,5,6.

(1) Algorithm

1. Append → Appends
   Extend() → Extend
   Insert() → Inserting
   Remove() → Removing
   pop() → pops
   clear() → clears all
   Index() → Indexing
   count() → counting
   sort() → Sorting
   Reverse() → Reversing
   copy() → Copying

code:

```
My list = [1,2,3,4,5]
Print ("Original list:")
My list.append(6)
Print ("After Append)
My list Extend
Print ("After Extend)
My list.Insert()
Pint ("Afr Insrt()
My.list (Remove
Print ("After Remove)
My.list.pop()
Print ("After pop")
Clear list.Clear()
print ("After clear)
My.list Index()
Print ("After Indexing")
My.list Count
Print ("Afr Counting
My.list Sort()
Print ("After Sorting")
My.list Reverse()
Print ("After Reversing")
My.list copying()
Print ("After Copying")
```

(2) Algorithm

1. Creating a tuple of Integers
2. Accessing Elements using Indexing
3. Accessing Element using slicing
4. Slicing with a step
5. Slicing in reverse order.

Experiment #		Student ID	
Date		Student Name	

Code:

```
My_tuple = (1,2,3,4,5,6,7,8,9)
Print(My tuple)
Print (My tuple [0])
Print (My tuple [-1])
Print (My tuple [2:5]) # (3,4,5)
Print (My tuple [5:]) # (6,7,8,9)
Print (My tuple [:3]) # (1,2,3)
Print (My tuple [::2]) (1,3,5,7,9)
Print (My tuple [::-1]) # (9,8,7,6,5,4,3,2,1)
```

1. Algorithm

creating a dictionary tuple of keys
Accessing a value
Modifying a value
Adding New pair
Removing a New pair.

Code:

```
My_dict [1,2]: 'a', (3,4): 'b', (5,6): 'c'}
Print (My_dict)
Print (My_dict [1,2])
print [My_dict [3,4]: 'x'
Print [My_dict]
print [My_dict['x']
Print [My_dict
del My_dict [(5,6)]
Print [My_dict]
```

Output: "a"

10 Algorithm

1. Initialize Empty String "Message"
2. Concatenate Hello on the input name
3. Add an Exclamatory Mark on End of Message
4. Return the message as the output
5. Call the function with "GURU" and "John"
6. Print the result.
   Print the result.

Code:

```
def greet(name):
    Message = " "
    Message = "Hello, " + name
    Message += "!"
    Return Message
Print (greet("John"))
```

Output:

Hello, GURU
Hello, John

Sample VIVA-VOCE Questions (In-Lab):

1. What is the difference between a list and a tuple in Python?
2. How does Python handle memory management?
3. Explain the purpose of the self-parameter in Python classes.
4. Describe how you can handle exceptions in Python. Provide an example.
5. What are decorators in Python and how are they used? Provide an example.

1A) In python, a list is a mutable collection of items that can be modified like as my_list = [1, 2, 3] my tuple = (1, 2, 3)

2A) Python handles memory management through automatic garbage collection which frees the developer from manually managing memory.

3A) Self parameter are:
Class My Class:
self.name = name.

4A) Exceptions using try-except block which allows you to catch try 2:5/0';

5A) · Decorators are special types of functions that can modify or extend the behaviour of another function.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post-Lab:

Implementation of Advanced Python Programs.

1. Write a function that takes two numbers and returns their sum, difference, produc quotient.
2. Implement a *recursive function* to calculate the factorial of a number.
3. Define a simple class **Person** with attributes *name* and *age*. Create **instances** of th Person class and print their attributes. Add a method *greet* to the Person class that a greeting message. Call the *greet* method on different instances and observe the
4. Create a subclass **Student** that inherits from the **Person** class. Add a new attribute *student_id* and a method *study* to the Student class. Create instances of **Student** a methods from both **Person** and **Student.**

- **Procedure/Program:**

① Code:

```
def Math-operations
if b=0:
raise zero division
    Error("Cannot divide by zero");
Sum_val = a+b.
diff-val = a-b
Prod-val = a*b
Quot-val = a/b;
return Sum_val
    diff-val
    Prod-val
    qivot-val
```

Output:

10

2

② Code:

```
def factorial(n):
if n=0 or n=1:
    return 1.
else:
return
    n* factorial(n-1)
```

Output:

5

120

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

(3) code:

```
class person:
    def init(self,name,age):
        self.name = name
        self.age = age
Person1 = person("John",30)
Person2 = person("Alice",25)

Print(f"person1: Name-{person1.name}
    {Age-{person1.age}")
Print(f"person2: Name-{person2.name}
    {Age-{person2.age}")

Person1.greet()
Person2.greet()
```

Output

Hello, My name is John I am 30 years old
Hello, My name is Alice I am 25 years old

(4) code:

```
Class person:
    def init(self,name,age
        self.name = name
        self.age = age
Class Student
    def init(self,name,age,stuid)
        super().init(name,age)
        self.stu_id = stu_id

Stu1 = Stu("Guru",20,"S101
Stu2 = Stu("Chaon"25"S103"

Student1.greet()
Student1.Study()
Student2.query()
Student2.Study()
```

Output

I am Guru I am 20 years Old. Studying with Student ID S101

Evaluator Remark (if Any):	Marks Secured: 50 out of 50
	Signature of the Evaluator with Date

Experiment #		Student ID	
Date		Student Name	

Experiment #2: Implement a Random Movement Reflex Agent.

Aim/Objective:

Implement a simple reflex agent for a vacuum cleaner.

Description:

Students will create a simple reflex agent to clean a grid-based environment. The agent will p
the status of the current cell and decide whether to clean, move, or do nothing.

Pre-Requisites:

The simplicity of the agent's decision-making process makes it an introductory exercise in a
intelligence, allowing students to understand the concept of reflex agents and their applica
autonomous systems.

Pre-Lab:

1. What is the purpose of implementing a reflex agent for a vacuum cleaner in this lab?

A) The purpose of implementing the reflexing agent for a vacuum is to create an Autonomous System.

2. What is a reflex agent, and how does it differ from other types of intelligent agents?

A) Simple intelligent Agent that agent to the Current State of the Environment Any information or memory differing from other types of agents like based reflex Agents.

3. Describe the grid-based environment in which the vacuum cleaner agent will operate. How is it structured?

The grid based Environment is 2D grid cells where Each Cell Represents a location in the Environment and the vaccum cleaner agent.

4. What are the possible states that a cell in the grid-based environment can have? How are these states represented?

The Possible States of a cell are '0' for Clean '1' for dirty and 2 for Obstacle.

5. What are the available actions that the vacuum cleaner agent can take in response to the current cell's status?

The Available Actions are:-

Move up
Move down
Move left
Move right
Suck
None.

Experiment #		Student ID	
Date		Student Name	

In-Lab:

The environment consists of a grid of squares(A,B), each of which can be either
dirty. The vacuum cleaner can perform the following actions:

- **Move Left:** Move one square to the left (if not at the leftmost edge).
- **Move Right:** Move one square to the right (if not at the rightmost edge).
- **Move Up:** Move one square up (if not at the top edge).
- **Move Down:** Move one square down (if not at the bottom edge).
- **Suck:** Clean the current square if it is dirty.

Write a program to create a vacuum cleaner agent that can clean all dirty squares in
efficiently. The agent should be able to sense its current position and the status (c
dirty) of the square it occupies.

Procedure/Program:

Algorithm

1. The grid Method is the Main Algorithm that cleans Entire grid.
2. It Repeadtly Senses the Current Square, Seeks if dirtty and Moves on a Adjacent Square
3. If not It finds the dirty.
4. It breaks and loop
5. Finally It prints "Grid is clean" and Exits.

Code:

```
class VacuumCleaner:
  def __init__(self, grid):
    self.grid = grid
    self.positions = [0,0]
  self.directions = [0,1], [0,-1], [1,0], [-1,0]
```

```
if all(all(cell == 'C' for cell in row) for row in grid):
    Print("Grid is Clean")
    Break.
if position[1] < len(grid[0])-1:
        position[1] += 1
else:
    position[0] += 1
    position[1] = 0
```

Data and Results: Output:

Cleaned Square at (0, 0)
Cleaned Square at (1, 1)
Grid is Clean

Analysis and Inferences:

1. The vacuum cleaner represents that vacuum cleaner agent
2. The 'Sense' Method checks if the current Square is dirty
3. The 'Suck' Method cleans the current Square is dirty.
- The 'Move' Method moves the vacuum cleaner to a new position based on the given direction.

Experiment #		Student ID	
Date		Student Name	

Sample VIVA-VOCE Questions (In-Lab):

1. How would you define a reflex agent in the context of artificial intelligence?
2. Explain the logic behind implementing random movement for a reflex agent.
3. What are the possible actions that the agent can take in response to the current cell's status?
4. How do you ensure that the agent's movements are indeed random and not following a predictable pattern?
5. How does the agent decide whether to clean, move, or do nothing based on the current cell status?
6. What are the potential challenges you might face when implementing a random movement agent, and how would you address them?

(1A) Reflex agent is in simple AI agent react to current environment.

(2A) Randomly selects one action from the available actions move up or left right.

(3A) Actions are: move up down left right suck.

(4A) Python's random module Eg: import random, Action: random.choice.

(5A) The agent checks the current cell's status if dirty it cleans if it moves randomly.

(6A) Getting stuck in loop, inefficient exploration solution add and randomness.

Post-Lab:

create a vacuum cleaner agent with three grid of squares (A,B and C) environment.

a) **Procedure/Program:**

Algorithm

1. Vaccum cleaner can sense the current grid
2. It suck any dirty squares
3. And move to next grid
4. Current grid and moves to the next one until the Entire Environment is clean.

code:

```
Class Vaccum cleaner:
    def __init__(self, grid):
        self.grid = grid
        self.position = 0
    def Sense(self):
        if (self.position == 0):
            return self.grid[0]
        elif
            self.position = 1,
            return self.grid[1];
        else:
            return self.grid[2]
    def Suck(self):
        current grid = self.Sense()
        if (current grid[i] == 'D'):
            Cunet grid[i], 'c'
        Print(cleaned Squane
            in grid)
    def clean_Environment(self):
        While True:
            self.Suck()
            if all (square == 'c' for
grid in self.grid for Square in
                grid):
                print("Environment is clean")
                $break
            self.Move()

grid-a = ['D', 'c', 'D']
grid-b = ['c', 'D', 'c']
grid-c = ['D', 'c', 'D']

Vaccum = vaccum cleaner
    (Environment)
Vaccum.clean_Environment()
```

Output:

vaccum is clean.

Experiment #		Student ID	
Date		Student Name	

b) Data and Results:

Cleaned Square in grid A
Cleaned Square in grid B
Cleaned Square in grid B
Cleaned Square in grid D
Cleaned Square in grid C

Environment is clean!

c) Analysis and Inferences:

It Sense returns the Current Stage of grid A
It Suck dirty Square of grid A
It moves vaccum cleaner to grid B
It Sense returns the Current Stage of grid B
It Suck It Suck dirty Square of grid B
It moves It vaccume cleaner to grid C
It Sense Return the Current Stage of grid C
It Suck It Suck dirty Squre of grid C
It move vaccum cleaner to grid C

Evaluator Remark (if Any):	Marks Secured: 50 out of 50
	Signature of the Evaluator with Date

Experiment # 3: Find the path to reach the target from a source node in given Graph

Aim/Objective:

Implementing BFS for finding the path from a source to the goal node in the graph.

Description:

Students will create a simple graph and traverse the graph using Breadth-first search. BFS is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighbor nodes. Then, it selects the nearest node and explores all the unexplored nodes.

Pre-Requisites:

a) Basic understanding of search algorithms.
b) Familiarity with Python programming language.
c) Knowledge of Graph traversing.

Pre-Lab:

1. What is Breadth-First Search (BFS)? Explain the basic idea behind BFS and how it explores a graph or a maze.

BFS is a Graph traversal Algorithm that Explores the Graph by Near Level Starting from the Given Source to destination

2. What data structure(s) are commonly used in BFS? Describe their purpose and how they help in implementing the algorithm efficiently.

A) Queue is a data Structure is Commonly used in BFS to keep the tracking node to visit And set or dict is used to keep track visited nodes

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

3. How can a grid-based maze be represented in a data structure? Discuss the options and which one you would choose for this implementation.

A) The grid maze can be represented by 2D Array list of lists, where each element represents a cell in maze with indicating walls(1) or path(0)

4. What are the steps involved in implementing BFS to solve a maze problem? Provide a high overview of the algorithm.

A) The steps are initializing a queue with the starting cell. Mark the starting cell. while queue is not empty.

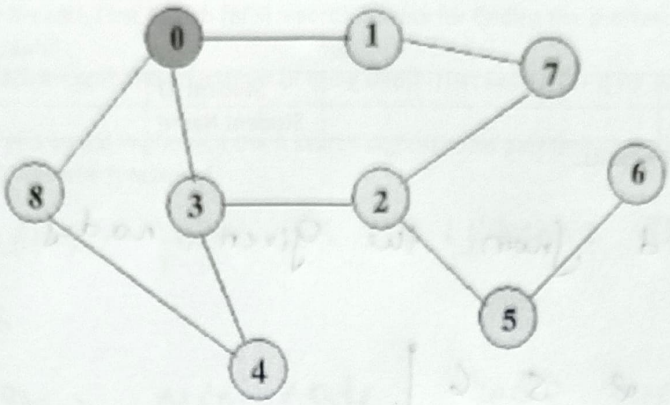
5. What is the purpose of marking visited cells in BFS? How can you keep track of visited efficiently during the traversal process?

A) Making visited cell prevents revisiting them avoiding infinite loops you can keep track visited cells using a seperated 2D array of lists.

Experiment #		Student ID	
Date		Student Name	

In-Lab:

Implement Breadth-First Search using Python and find a path from *node 0* to *node 6* for the graph below using BFS.



Procedure/Program:

Algorithm

1. Arrange the given nodes
2. Check whether the satisfy the BFS
3. Check the Node from 0 to 6
4. Define the graph from the Adjacency list.

Code:

```
from Collections import deque
Graphs {
  0: [1,2],
  1: [0,3,4],
  2: [0,5],
  3: [1],
  4: [1],
  5: [2][5],
  6:
def bfs(graph, start, goal):
    queue = deque([[start]])
    visited = set
    while queue:
        path = queue.popleft()
        node = path[-1]
    visited.add(node)
    for neighbour in graph.get(node,[]
        new_path = list(path)
        new_path = list(path)
        new_path.append(neighbour)
        queue.append(new_path)
    return none
Output:
if path:
    Print("path found:", path)
else:
    Print("No path found")
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:**

Path found from the given nodes

Path: [0 2 5 6]

- **Analysis and Inferences:**

1. The graph is represented as an adjacency list where each key is a node and its value is a list of its neighbors.
2. The bfs implements the Breadth-first-Search
3. The queue to keep track of nodes to visit and set to keep track of visited nodes.
4. The Algorithm starts by adding the start node to the Queue.
5. If the goal is not found, the Algorithm returns the path from the start node to the goal node.

Experiment #		Student ID	
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Sample VIVA-VOCE Questions (In-Lab):

1. What are the different algorithms available for finding the shortest path in a graph, and how do they differ?
2. Explain the working of Dijkstra's algorithm and discuss its time complexity.
3. How does the Breadth-First Search (BFS) algorithm work for finding the shortest path in an unweighted graph?
4. What are the advantages and limitations of using Depth-First Search (DFS) for pathfinding in a graph?
5. Describe how you would implement the A search algorithm for pathfinding in a graph, and explain the role of the heuristic function. *

1A) Shortest path is Dijkstra's and BFS and DFS

2A) Start Min node Ends with Max node.

3A) BFS is a Algorithm it starts a node to End with Queue property

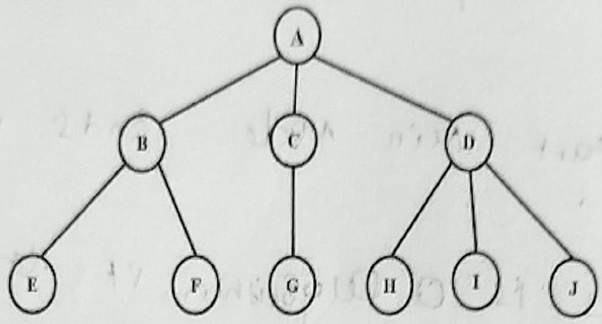
4A) DFS is a Algorithm it starts a node to End with Stack property

5A) A Search in Algorithm start goal if node in path in or node, goal

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post-Lab:

Implement DFS for the given tree.



- **Procedure/Program:**

Algorithm

Arrange the given nodes
2. Check whether the Satisfy the BFS
3. Check the Node from 0 to 6
4. Define the graph from Adjacency list.

Code:

```
Class Graph:
def __init__(self):
self.graph = {
'A' : [B, C, D],
B = [E, F]
C = ['G']
D : ['H', 'I', 'J'],
E : [],
F : []
G : [],
H : [],
I : []
J : []
def dfs(self, start_node):
    visited = set()
self.dfs_helper(start_node, visited)
def _dfs_helper(self, node, visited):
    visited.add(node)
    Print(node, end=" ")
    for neighbor in self.graph[node]:
    if neighbour not in visited:
    self._dfs_helper(neighbour, visited)
graph = Graph()
Print("DFS traversal starting from Node A:")
graph.dfs('A')
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

- **Data and Results:** output:

DFS traversal starting from Node A:

A
B
E
F
C
G
D
H
I
J

- **Analysis and Inferences:**

1. The graph is represented as an adjacency list where each key is a node and its value is a list of neighbour
2. the method to perform the DFS traversal
3. A start node as input and calls the method to perform the recursive DFS traversal.
4. The DFS traversal starts from node A and visits all nodes in the graph.

Evaluator Remark (if Any):	Marks Secured: 50 out of 50
	Signature of the Evaluator with Date