Lab 1: Solving Problems using AI (Python Program)

```
secret = 7
guess = int(input("Guess the number: "))

if guess == secret:
    print("Correct Guess!")
else:
    print("Wrong Guess!")
```

Lab 3: Propositional Logic and Reasoning (Python)

```
P = True
Q = False

print("P AND Q =", P and Q)
print("P OR Q =", P or Q)
print("NOT P =", not P)

result = (not P) or Q
print("P implies Q =", result)
```

Lab 4: Expert System in Prolog

```
marks(rahul, 75).

result(Student, pass) :-
    marks(Student, Marks),
    Marks >= 50.

result(Student, fail) :-
```

```
    marks(Student, Marks),
    Marks < 50.
```

Lab 5: Uninformed Search – Depth First Search (DFS) in Python

```
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': [],
    'F': []
}

visited = set()

def dfs(node):
    if node not in visited:
        print(node, end=" ")
        visited.add(node)
        for neighbour in graph[node]:
            dfs(neighbour)

dfs('A')
```

Python Program – A* Search Algorithm

```
# Simple A* Search Algorithm

graph = {
    'A': {'B': 1, 'C': 3},
    'B': {'D': 1, 'E': 5},
    'C': {'F': 2},
    'D': {},
    'E': {},
    'F': {}
}

# Heuristic values
h = {
    'A': 6,
    'B': 4,
    'C': 4,
    'D': 0,
    'E': 0,
    'F': 0
}

open_list = ['A']
closed_list = []

g = {'A': 0}
parent = {'A': 'A'}

goal = 'D'

while open_list:
    n = open_list[0]
    for v in open_list:
        if g[v] + h[v] < g[n] + h[n]:
            n = v

    if n == goal:
        path = []
        while parent[n] != n:
            path.append(n)
            n = parent[n]
        path.append('A')
        path.reverse()
        print("Path found:", path)
        break

    for m in graph[n]:
        cost = graph[n][m]
        if m not in open_list and m not in closed_list:
            open_list.append(m)
            parent[m] = n
            g[m] = g[n] + cost

    open_list.remove(n)
    closed_list.append(n)
```

Control structure in prolog

```
% Control Structure in Prolog (If-Then-Else)
```

```
number_type(X) :-
    ( X > 0 ->
        write('Positive number')
    ; X < 0 ->
        write('Negative number')
    ;
        write('Zero')
    ).
```

RECURSION IN PROLOG

```
% Recursion in Prolog - Factorial Program

factorial(0,1).
factorial(N,F) :-
    N > 0,
    N1 is N - 1,
    factorial(N1,F1),
    F is N * F1.
```

```
# Simple Supervised Learning (without package)

# Training data
X = [1, 2, 3, 4, 5]
Y = [2, 4, 6, 8, 10]

# Learning (simple rule)
def train(x):
    return x * 2
```

```
# Testing
test = 6
prediction = train(test)

print("Prediction:", prediction)
```

# Simple Bayesian Learning Example

```

# Prior probabilities
p_spam = 0.6
p_not_spam = 0.4

# Likelihood
p_offer_given_spam = 0.7
p_offer_given_not_spam = 0.2

# Evidence
p_offer = (p_offer_given_spam * p_spam) + (p_offer_given_not_spam * p_not_spam)

# Bayes Theorem
p_spam_given_offer = (p_offer_given_spam * p_spam) / p_offer

print("Probability the mail is spam:", p_spam_given_offer)
```

# Simple Clustering (Unsupervised Learning)

```

# Data points
data = [1, 2, 3, 10, 11, 12]
```

```
# Initial clusters
cluster1 = []
cluster2 = []

# Simple clustering rule
for x in data:
    if x < 6:
        cluster1.append(x)
    else:
        cluster2.append(x)
print("Cluster 1:", cluster1)
print("Cluster 2:", cluster2)
```

# Simple Reinforcement Learning Example

```
reward = 0
state = 0

for step in range(5):
    action = "move"

    if state < 3:
        reward = reward + 1
        state = state + 1
    else:
        reward = reward - 1

print("Final State:", state)
print("Total Reward:", reward)
```