1) PEAS and State Description for 'Automobile Driver Agent'

PEAS:

Performance Measure: Safety, reaching destination, fuel efficiency, obeying traffic laws, passenger comfort.

Environment: Roads, traffic signals, other vehicles, pedestrians, weather.

Actuators: Steering, accelerator, brakes, indicators.

Sensors: Cameras, GPS, speedometer, radar, LIDAR, proximity sensors.

State Description:

An automobile driver agent is an intelligent system capable of perceiving road conditions, interpreting traffic signals, and taking appropriate driving actions to safely and efficiently navigate from a starting point to a destination.

2) PEAS and State Description for 'Online English Tutor'

PEAS:

Performance Measure: Student improvement, engagement, test scores, session completion.

Environment: Students, online platform, teaching materials.

Actuators: Display screen, audio output, message/chat.

Sensors: Keyboard input, speech recognition, webcam (optional), quiz results.

State Description:

An online English tutor is a software agent that interacts with students over the internet to improve their English language skills by providing lessons, correcting grammar, and evaluating progress.

3) Adversarial Search using Min-Max Algorithm

Code:

```
def minmax(depth, nodeIndex, isMax, scores, targetDepth):
```

```
    if depth == targetDepth:
        return scores[nodeIndex]
    if isMax:
        return max(minmax(depth+1, nodeIndex*2, False, scores, targetDepth),
                   minmax(depth+1, nodeIndex*2 + 1, False, scores, targetDepth))
    else:
        return min(minmax(depth+1, nodeIndex*2, True, scores, targetDepth),
                   minmax(depth+1, nodeIndex*2 + 1, True, scores, targetDepth))


scores = [3, 5, 6, 9, 1, 2, 0, -1]
print("Optimal value:", minmax(0, 0, True, scores, 3))
```

4) BFS Program (Python)

Code:

```
from collections import deque
def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        vertex = queue.popleft()
        if vertex not in visited:
            print(vertex, end=" ")
            visited.add(vertex)
            queue.extend(set(graph[vertex]) - visited)
graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
bfs(graph, 'A')
```

5) DFS Program (Python)

Code:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=" ")
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
graph = {'A': ['B', 'C'], 'B': ['D', 'E'], 'C': ['F'], 'D': [], 'E': ['F'], 'F': []}
dfs(graph, 'A')
```

6) PEAS and State Description for 'Medical Diagnosis System'

PEAS:

Performance Measure: Accuracy of diagnosis, patient recovery, speed of diagnosis.

Environment: Patient symptoms, historical medical records, test results.

Actuators: Display diagnosis, suggest treatment, alerts.

Sensors: Inputs from user, data from medical tests and records.

State Description:

A medical diagnosis system is an intelligent agent that analyzes patient symptoms and medical data to suggest possible diagnoses and treatment options, aiding healthcare professionals in decision-making.

7) Hill Climbing Algorithm (Python)

Code:

```
def hill_climb(function, x_start, max_iterations, step_size):
    x = x_start
    for _ in range(max_iterations):
        next_x = x + step_size
        if function(next_x) > function(x):
            x = next_x
        else:
            break
    return x
def f(x):
    return -x**2 + 4
result = hill_climb(f, x_start=0, max_iterations=100, step_size=0.1)
print("Local maximum at x =", result, "with value =", f(result))
```