**Robot\_vacuum**

# **Task 1:**

1. **Môi trường (environment):**
   * Dùng một **mảng 2D** (numpy array) để biểu diễn phòng (clean/dirty).
   * Vị trí của robot là một cặp tọa độ (x, y).
   * Có thể cho kích thước phòng nhỏ (2×2, 3×3) để dễ quan sát.
2. **Percept:**
   * Agent sẽ nhận được (vị trí hiện tại, trạng thái ô đó).
3. **Action:**
   * 'Suck': làm sạch ô hiện tại.
   * 'Up', 'Down', 'Left', 'Right': di chuyển robot.
   * 'NoOp': không làm gì.
4. **Performance measure:**
   * +10 nếu làm sạch thành công một ô bẩn.
   * -1 cho mỗi bước di chuyển.
   * Có thể -1 cho mỗi bước thời gian để khuyến khích hiệu quả.
5. **Chạy mô phỏng:**
   * Cho vòng lặp tối đa max\_steps.
   * Sau mỗi bước: gọi agent → nhận hành động → cập nhật môi trường → tính điểm.
   * Nếu tất cả ô sạch → dừng.
   * Nếu verbose=True: in trạng thái phòng và vị trí robot.

**CODE TASK 1**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**Giải thích:**

1. **Khởi tạo:** tạo mảng room với **"Dirty"/"Clean**" theo **dirt\_prob**, chọn vị trí bắt đầu **agent\_pos** ngẫu nhiên**.**
2. **Vòng lặp chạy:**

* Gọi **percept()** → trả vị trí hiện tại và trạng thái ô đó**.**
* Gọi action = agent(percept). **Agent chỉ thấy percept này.**
* **Gọi execute(action):**
  + Nếu **Suck** và ô bẩn → đổi thành **Clean**, +10 điểm.
  + **Nếu di chuyển:**
    - nếu hợp lệ (không vượt biên) → cập nhật vị trí, -1 điểm.
    - nếu đi vào tường (invalid) → -2 điểm (bumper penalty), vị trí không thay đổi.
  + Mỗi bước đều -1 điểm thời gian.
* Kiểm tra **is\_done()** → nếu tất cả **Clean** hoặc **steps >= max\_steps** thì dừng.

1. **Trả performance**: tổng điểm tích lũy (thưởng - phạt).

**Thiết kế performance measure**

* +10 cho mỗi Suck trên ô bẩn khuyến khích dọn.
* -1 mỗi bước khuyến khích tiết kiệm hành động/đi thẳng vào mục tiêu.
* -2 khi va chạm (bumper) để phạt hành động cố tình đụng tường.

**Debugging & kiểm thử (rất quan trọng)**

* **Verbose:** bật **verbose=True** trong run để in từng bước — cần cho debug.
* **Seed:** dùng tham số seed để tái lập thử nghiệm.
* **Trường hợp kiểm tra:**
  + Tất cả ô **Clean** ban đầu → **agent** nên dừng sớm (NoOp or steps small).
  + Tất cả ô **Dirty** ban đầu → kiểm tra agent có thể dọn hết (nếu logic đúng).
  + Agent cố gắng đi ra ngoài → xem có phạt bumper không.
* **Kiểm tra agent cũ**: nếu agent ban đầu viết để nhận **(pos,status)** thì đảm bảo environment **percept()** trả đúng 2-tuple (Task 1 yêu cầu). Tránh đổi percept thành 3-tuples ở Task 1.

# **Task 2: Implement a Simple Reflex Agent [10 Points]**

**Viết một simple reflex agent hoạt động như sau:**

* Đi **loanh quanh ngẫu nhiên** trong môi trường.
* Nếu gặp tường (bumper sensor → va chạm) thì không đi xuyên qua, mà chọn hướng khác.
* Nếu ô hiện tại bẩn → "Suck".

Agent **không được truy cập trực tiếp vào biến của environment** (chỉ dùng percepts được cung cấp).

Hàm agent phải có cùng function signature như **simple\_randomized\_agent** đã cho trong notebook.

**CODE TASK 2:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a computer program

AI-generated content may be incorrect.**

**Giao diện hàm (function signature) simple\_reflex\_agent:**

* **Input (percept):** một giá trị percept do environment trả về. Có hai biến thể percept thường gặp:
  1. **Percept dạng đầy đủ (khuyến nghị cho Task 2):** một 3-tuple (position, status, walls)
     + position = (row, col) — vị trí hiện tại của robot.
     + status = "Clean" hoặc "Dirty" — trạng thái ô tại vị trí hiện tại.
     + walls = dictionary hoặc struct báo tường (bumper) có dạng {"Up": True/False, "Down":..., "Left":..., "Right":...} (True = có tường/khoá).
  2. **Percept dạng đơn giản (Task 1 chuẩn): một 2-tuple (position, status)**
     + Trường hợp này agent không biết tường và chỉ phản ứng với dirt; environment xử lý va chạm khi agent chọn hướng dẫn tới tường (phạt, không di chuyển).
* **Output (action):** một chuỗi (string) trong tập {"Suck", "Up", "Down", "Left", "Right", "NoOp"}.

Ghi vào Word: nêu rõ percept bạn dùng trong báo cáo (rất quan trọng để đọc người chấm hiểu agent).

**Luồng quyết định:**

Nhận percept từ environment (position, status[, walls]).

Kiểm tra trạng thái ô hiện tại:

* Nếu status == "Dirty" → trả "Suck". (Hành động hút ngay lập tức; không xét hướng di chuyển.)
* Nếu status == "Clean" → đi bước 3.

Khi ô sạch, xây danh sách hành động di chuyển khả dĩ:

* Danh sách gốc: ["Up","Down","Left","Right"].
* Nếu percept chứa walls, loại bỏ những hướng mà walls[direction] == True (tức hướng đó bị chặn).
* Thêm tùy chọn "NoOp" (đứng yên) vào danh sách để agent có thể không di chuyển.

Chọn hành động:

* Chọn ngẫu nhiên (uniform) một hành động từ danh sách đã lọc.
* Trả về hành động đó cho environment.

Environment thực hiện action: nếu move hợp lệ → cập nhật vị trí; nếu move dẫn tới tường (và agent không biết walls) → environment phạt (bumper). Nếu action là Suck và ô thực sự bẩn → môi trường làm sạch ô và thêm reward.

Lặp lại cho đến khi điều kiện dừng (tất cả ô sạch hoặc hết bước).

**Xử lý “bumper sensor” (tường/chướng ngại)**

* **Nếu environment cung cấp walls**: agent **chủ động** tránh chọn hướng bị chặn → KHÔNG va chạm.
* **Nếu environment không cung cấp walls**: agent vẫn chọn random; environment phải **xử lý va chạm** (không di chuyển và/hoặc trừ điểm). Task yêu cầu agent *reacts to bumper*, nên tốt nhất môi trường trả walls và agent dùng nó để tránh bump.

# **Task 3: Implement a Model-Based Reflex Agent [20 Points]**

Viết một model-based reflex agent có trí nhớ nội bộ (internal state).

**Yêu cầu:**

* Agent phải ghi nhớ trạng thái đã quan sát của từng ô (sạch hay bẩn).
* Nếu ô hiện tại bẩn → "Suck".
* Nếu ô hiện tại sạch → di chuyển sang ô khác chưa biết hoặc còn bẩn.
* Nếu tất cả ô đều đã được biết và đều sạch → "NoOp" (dừng).

Agent chỉ được nhận **percept** từ environment (position, status) → không được truy cập trực tiếp vào room.

**CODE TASK 3:**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Luồng hoạt động chính của hàm agent**

1. **Khởi tạo model nội bộ:**
   * Agent tạo ra một cấu trúc dữ liệu (ví dụ ma trận 2D hoặc dictionary) để lưu trạng thái của từng ô: "Dirty", "Clean", hoặc "Unknown".
   * Ban đầu, tất cả ô đều "Unknown" cho đến khi agent ghé qua.
2. **Nhận percept từ environment:**
   * Lấy position (tọa độ ô hiện tại).
   * Lấy status ("Dirty" hoặc "Clean").
   * Nếu có, lấy thông tin walls để biết hướng nào bị chặn.
3. **Cập nhật model (update state):**
   * Đánh dấu ô hiện tại trong bộ nhớ là "Clean" hoặc "Dirty" dựa vào percept.
   * Nếu percept báo "Dirty", agent sẽ xử lý ngay bằng hành động "Suck".
4. **Xác định mục tiêu tiếp theo (goal selection):**
   * Nếu có ô "Dirty" đã biết trong bộ nhớ → chọn đi đến đó.
   * Nếu không còn "Dirty" đã biết nhưng còn ô "Unknown" → chọn đi khám phá ô chưa biết.
   * Nếu tất cả đã "Clean" → có thể chọn "NoOp".
5. **Lập kế hoạch di chuyển (path planning):**
   * Agent tìm đường đi ngắn nhất từ vị trí hiện tại đến ô mục tiêu (thường dùng BFS).
   * Nếu ô bị tường/obstacle, agent sẽ cập nhật model để tránh nó.
6. **Chọn hành động tiếp theo:**
   * Nếu vị trí hiện tại đang "Dirty" → trả "Suck".
   * Nếu không → lấy bước đầu tiên trong đường đi ngắn nhất (ví dụ "Up", "Down", …).
   * Nếu không tìm thấy đường nào → chọn hành động ngẫu nhiên như fallback.
7. **Environment thực hiện action:**
   * Robot di chuyển hoặc hút bụi; environment cập nhật trạng thái thực tế.
   * Agent sẽ nhận percept mới ở vòng sau và lặp lại toàn bộ quá trình.

**Tóm gọn**: Hàm luồng chính của **model-based reflex agent** hoạt động theo 3 pha lặp lại:  
(**1) nhận percept → (2) cập nhật model nội bộ → (3) quyết định action dựa trên model (Suck nếu bẩn, di chuyển theo kế hoạch nếu sạch).**

# **Task 4: Simulation study [30 Points].**

Chạy so sánh hiệu suất (performance measure) của 3 agent:

1. **Randomized Agent (Task 1 agent ngẫu nhiên).**
2. **Simple Reflex Agent (Task 2).**
3. **Model-based Reflex Agent (Task 3).**

Trên các môi trường có kích thước: 5×5, 10×10, 100×100.

Mỗi kích thước → chạy 100 lần ngẫu nhiên.

Ghi kết quả trung bình vào bảng.

Vẽ biểu đồ so sánh hiệu suất.

Viết vài dòng thảo luận kết quả.

**CODE TASK 4:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A computer screen shot of a black and white screen

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A black screen with white text

AI-generated content may be incorrect.**

**A graph with a number of blue and white bars

AI-generated content may be incorrect.**

**Xây dựng hàm evaluate\_agent:**

* **Input:** một hàm agent, kích thước phòng (width, height), số lần chạy (runs), và số bước tối đa (max\_steps).
* **Quá trình:**
  + Tạo môi trường mới nhiều lần (mỗi run một môi trường ngẫu nhiên).
  + Chạy agent trong môi trường đó cho đến khi dừng.
  + Ghi lại điểm số (performance) thu được.
* **Output:** giá trị trung bình performance trên nhiều lần chạy → đại diện cho hiệu quả của agent.

**Chọn danh sách agent:**

* Tạo dictionary lưu tên agent → hàm agent.
* **Ví dụ:** { "Randomized": random\_agent, "Simple Reflex": simple\_reflex\_agent, "Model-based Reflex": model\_based\_agent\_fn }.

**Chọn kích thước môi trường để so sánh:**

* Nhỏ (5×5), trung bình (10×10), lớn (100×100).
* Điều này kiểm tra agent trong cả phòng nhỏ dễ dọn và phòng lớn phức tạp.

**Chạy thí nghiệm:**

* Với mỗi kích thước phòng, gọi evaluate\_agent cho từng agent.
* Lưu kết quả (trung bình performance).

**Biểu diễn kết quả:**

* In ra bảng kết quả (performance trung bình).
* Vẽ biểu đồ (bar chart hoặc line chart) để trực quan so sánh.

**Luồng hoạt động của Task 4 (chi tiết):**

1. **Khởi động:** chọn một agent (ví dụ Simple Reflex).
2. **Lặp qua số runs:**

* Sinh ngẫu nhiên môi trường (dirty/clean layout khác nhau).
* Cho agent chạy trong môi trường này.
* Sau khi kết thúc, lưu lại performance score.

1. **Tính trung bình:** lấy mean của tất cả performance scores.
2. **Lặp lại cho tất cả agent khác**.
3. **So sánh:** đưa tất cả kết quả vào bảng/biểu đồ.

**Ý nghĩa:**

**Randomized agent**: thường có performance thấp, đặc biệt ở phòng lớn (do di chuyển lung tung).

**Simple Reflex agent:** tốt hơn random, tránh bump, dọn hiệu quả trong phòng nhỏ nhưng vẫn kém trong phòng lớn vì không có trí nhớ.

**Model-based Reflex agent**: tận dụng bộ nhớ để tránh lặp lại, hiệu quả hơn hẳn khi phòng có nhiều ô.

* Task 4 là quá trình tạo vòng lặp đánh giá → chạy nhiều lần → lấy trung bình → so sánh 3 agent bằng bảng và biểu đồ.

# **Task 5: Robustness of the agent implementations [10 Points].**

**Nhiệm vụ:** Mô tả bằng lời cách mà 3 loại agent (Randomized, Simple Reflex, Model-based Reflex) sẽ hoạt động khi gặp các tình huống phức tạp.

**A screenshot of a computer

AI-generated content may be incorrect.**

# **Advanced task: Imperfect Dirt Sensor**

**Thay đổi simulation environment để chạy thử nghiệm với tình huống:**

* Dirt sensor có 10% khả năng đọc sai (khi ô bẩn → báo sạch, khi ô sạch → báo bẩn).
* Chạy thử nghiệm để quan sát ảnh hưởng tới performance của 3 loại agent trước đó (Randomized, Simple Reflex, Model-based Reflex).
* Lưu ý: với sensor không hoàn hảo, model-based reflex agent có thể không dọn hết phòng → cần đo performance theo trade-off giữa energy cost và số ô chưa sạch.

**Thiết kế và implement giải pháp cải tiến cho model-based agent để làm việc tốt hơn khi sensor không hoàn hảo.**

* Sau đó chạy thử nghiệm để so sánh kết quả.

**CODE:**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Cách thực hiện trong môi trường**

1. **Thêm tham số noise (xác suất lỗi sensor):**
   * Ví dụ: noise\_prob = 0.1 (tức 10% khả năng sensor báo sai).
2. **Khi environment tạo percept:**
   * Nếu ô hiện tại là "Dirty", sensor có thể báo "Clean" với xác suất noise.
   * Nếu ô hiện tại là "Clean", sensor có thể báo "Dirty" với xác suất noise**.**
3. **Kết quả:** percept mà agent nhận được không chắc chắn phản ánh đúng trạng thái thật.

**Luồng hoạt động của agent trong tình huống này**

1. **Nhận percept có thể sai:**
   * Ví dụ percept báo "Dirty", nhưng thực tế ô đó sạch.
2. **Quyết định dựa trên xác suất:**
   * Agent không nên tin tuyệt đối vào sensor**.**
   * Một chiến lược cơ bản:
     + Nếu sensor báo "Dirty" → vẫn ưu tiên thực hiện "Suck", vì chi phí hút nhầm thường nhỏ hơn bỏ sót bụi thật**.**
     + Nếu sensor báo "Clean" → agent có thể tin nhưng cần thêm cơ chế kiểm tra lại (ví dụ đi vòng và quay lại sau để xác nhận).
3. **Cập nhật bộ nhớ (nếu model-based):**
   * Agent duy trì một model nội bộ chứa niềm tin (belief) về mỗi ô, thay vì trạng thái chắc chắn.
   * **Ví dụ:** mỗi ô có một xác suất là Dirty, và agent cập nhật dần dần khi nhận percept mới.
4. **Ra quyết định:**
   * Nếu xác suất một ô là Dirty vượt quá ngưỡng (ví dụ > 0.6) → chọn "Suck".
   * Nếu thấp → chọn hành động di chuyển sang ô khác.

**Tóm tắt**: Trong Advanced Task, ta giả định sensor báo trạng thái sạch/bẩn có sai số. Mỗi lần nhận percept, agent có khả năng nhận thông tin sai so với thực tế. Vì vậy, agent phải điều chỉnh chiến lược: thường vẫn ưu tiên hút khi sensor báo “Dirty” để tránh bỏ sót, và có thể dùng bộ nhớ hoặc xác suất để ra quyết định hợp lý hơn. Kết quả là hiệu năng giảm so với trường hợp sensor hoàn hảo, nhưng agent model-based có thể giảm thiểu ảnh hưởng nhờ tận dụng trí nhớ.

# **More Advanced Implementation**

1. Hiện thực **môi trường có obstacles (ngẫu nhiên**) và giải thích cách agent bị bumper khi cố đi vào obstacle.

**Code:**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**A screen shot of a computer program

AI-generated content may be incorrect.**A screen shot of a computer program

AI-generated content may be incorrect.A screen shot of a computer program

AI-generated content may be incorrect.

**A screenshot of a computer program

AI-generated content may be incorrect.**

1. **Khởi tạo lưới (grid):**
   * Ngoài trạng thái "Clean" và "Dirty", một số ô được đánh dấu "Obstacle".
   * Obstacles được chọn ngẫu nhiên với xác suất định trước (ví dụ 10%).
2. **Percept:**
   * Agent vẫn nhận được vị trí và trạng thái hiện tại của ô.
   * Nếu agent cố gắng di chuyển vào obstacle → percept lần sau vẫn là cùng một ô (robot không di chuyển), và environment sẽ trừ điểm (penalty).
3. **Thực thi hành động:**
   * Với "Suck": chỉ có tác dụng nếu ô hiện tại không phải obstacle và đang "Dirty".
   * Với hành động di chuyển (Up, Down, …): nếu hướng đó là obstacle hoặc ra ngoài biên → robot không di chuyển và bị phạt.

**Luồng hoạt động chính**

**Bước 1:** Agent đưa ra hành động (ví dụ "Right").

**Bước 2:** Environment kiểm tra:

* + Nếu ô bên phải là obstacle → robot vẫn ở nguyên vị trí, performance giảm do bumper.
  + Nếu không → robot di chuyển sang ô đó.

**Bước 3:** Environment cập nhật percept (position mới, trạng thái ô mới).

**Bước 4:** Lặp lại cho đến khi tất cả ô sạch (ngoại trừ obstacle) hoặc hết bước.

**Tác động đến agent**

Random Agent / Simple Reflex Agent: thường xuyên đâm vào obstacles → bị phạt nhiều → hiệu năng giảm rõ rệt.

Model-based Agent: có thể ghi nhớ vị trí obstacles, tránh thử đi lại → cải thiện hiệu năng đáng kể.

**Tóm tắt:** Trong advanced task này, môi trường được mở rộng bằng cách thêm obstacles. Đây là những ô robot không thể đi vào. Nếu agent cố gắng di chuyển vào một obstacle, environment sẽ phát hiện bumper và trừ điểm, đồng thời giữ robot tại chỗ. Điều này khiến hiệu năng của các agent đơn giản giảm mạnh, trong khi agent model-based có lợi thế vì có thể ghi nhớ vị trí obstacles và tránh lặp lại sai lầm.

1. **Viết agent khám phá/phủ kín** (closest unchecked/uncleaned) — dùng tìm đường BFS để di chuyển đến ô cần kiểm tra (gần nhất). Agent này hoạt động khi không biết kích thước/phạm vi ban đầu — nó xây bản đồ dần dần.

**Code:**

A screen shot of a computer program

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Luồng hoạt động chính**

1. **Khởi tạo bộ nhớ nội bộ (model):**
   * Agent tạo một bản đồ rỗng (toàn bộ ô là "Unknown").
   * Vị trí ban đầu được đánh dấu theo percept đầu tiên.
2. **Nhận percept từ environment:**
   * Agent biết được trạng thái ô hiện tại ("Dirty" hoặc "Clean").
   * Nếu là "Dirty" → hành động ngay "Suck" và cập nhật bản đồ.
3. **Cập nhật model:**
   * Đánh dấu ô hiện tại trong bản đồ là "Clean" sau khi hút, hoặc "Clean" nếu percept báo sạch.
   * Nếu agent gặp bumper (cố đi vào tường/obstacle), ô đó được ghi nhớ là "Obstacle".
4. **Xác định mục tiêu tiếp theo:**
   * Nếu trong bản đồ còn ô "Dirty" đã biết → chọn ô Dirty gần nhất.
   * Nếu không còn Dirty nhưng còn "Unknown" → chọn ô chưa kiểm tra gần nhất.
   * Nếu tất cả đã "Clean" → có thể dừng hoặc chọn NoOp.
5. **Lập kế hoạch di chuyển (Path planning):**
   * Dùng thuật toán tìm đường (thường là **BFS**) để tính đường đi ngắn nhất từ vị trí hiện tại đến ô mục tiêu (Dirty/Unknown).
   * Bỏ qua các ô đã đánh dấu là "Obstacle".
6. **Chọn hành động tiếp theo:**
   * Nếu ô hiện tại Dirty → "Suck".
   * Nếu không → lấy bước đầu tiên trong đường đi đã tính (ví dụ "Up", "Left", …).
7. **Environment thực hiện action** → trả percept mới → quay lại bước 2.

**Ví dụ minh họa**

* Phòng 3×3, robot bắt đầu ở (0,0).
* Bản đồ nội bộ ban đầu: tất cả ô "Unknown".

1. Percept: (0,0) Dirty → agent "Suck" → (0,0) được ghi "Clean".
2. Agent tìm trong bản đồ: còn nhiều ô "Unknown" → chọn ô gần nhất, ví dụ (0,1).
3. BFS tìm đường đi: từ (0,0) đến (0,1) = "Right".
4. Agent đi "Right". Nếu ô (0,1) Dirty → "Suck". Nếu Clean → đánh dấu "Clean", rồi tìm ô Unknown tiếp theo.
5. Quá trình lặp lại cho đến khi toàn bộ lưới được khám phá (trừ obstacles).

**Ưu điểm**

* Bảo đảm **khám phá toàn bộ môi trường** thay vì đi ngẫu nhiên.
* Tránh obstacles nhờ ghi nhớ vị trí bumper.
* Tối ưu hơn Simple Reflex Agent vì có **chiến lược rõ ràng** (đi tới gần nhất).

**Nhược điểm**

* Cần bộ nhớ để lưu bản đồ → tốn tài nguyên hơn.
* Nếu môi trường rất lớn, BFS nhiều lần có thể tốn thời gian tính toán.

**Tóm tắt**: Agent khám phá/phủ kín hoạt động bằng cách lưu lại một bản đồ nội bộ. Mỗi lần nhận percept, agent cập nhật trạng thái ô hiện tại và nếu gặp bumper thì đánh dấu obstacle. Sau đó agent luôn tìm ô Dirty hoặc Unknown gần nhất trong bản đồ và sử dụng BFS để tìm đường đi đến đó. Khi tới ô Dirty, agent sẽ Suck ngay lập tức. Chiến lược này giúp agent bao phủ toàn bộ môi trường một cách hệ thống và hiệu quả hơn nhiều so với việc di chuyển ngẫu nhiên.

1. Mô tả & cấp khung code khởi động **cho utility-based agent** trong môi trường có xác suất ô bị bẩn lại — cho ý tưởng học xác suất và chiến lược greedy làm tăng utility.

**Code:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**A black background with white text

AI-generated content may be incorrect.**