Viết 1 prompt yêu cầu ChatGPT giúp bạn tóm tắt kiến thức về cấu trúc Queue sao cho dễ nhớ để ôn thi:

-Hãy đóng vai một gia sư lập trình giàu kinh nghiệm.

Tôi là sinh viên đang ôn thi môn Cấu trúc dữ liệu và Giải thuật và cần một bản tóm tắt súc tích, dễ nhớ về cấu trúc dữ liệu Queue.

Vui lòng giúp tôi soạn một bản tóm tắt theo cấu trúc sau:

Ví dụ đời thực: Bắt đầu bằng một ví dụ cực kỳ đơn giản và trực quan để tôi hình dung (ví dụ: xếp hàng chờ thanh toán).

Nguyên tắc cốt lõi: Giải thích rõ ràng nguyên tắc FIFO (First-In, First-Out) - "Vào trước, ra trước".

Các phép toán chính: Liệt kê và giải thích ngắn gọn các hoạt động cơ bản nhất của Queue, bao gồm:

Enqueue: Thêm một phần tử vào cuối hàng đợi.

Dequeue: Lấy và xóa một phần tử khỏi đầu hàng đợi.

Peek (hoặc Front): Xem phần tử ở đầu hàng đợi mà không xóa nó.

isEmpty: Kiểm tra xem hàng đợi có rỗng không.

isFull/Size: Kiểm tra xem hàng đợi có đầy không hoặc trả về kích thước hiện tại.

Cách cài đặt: So sánh ngắn gọn 2 cách cài đặt phổ biến là dùng mảng (array) và dùng danh sách liên kết (linked list), chỉ ra ưu và nhược điểm chính của mỗi cách khi cài đặt Queue.

Ứng dụng thực tế: Nêu 3-4 ứng dụng quan trọng trong thực tế (ví dụ: quản lý tác vụ trong hệ điều hành, thuật toán tìm kiếm theo chiều rộng BFS, hàng đợi in ấn).

Yêu cầu quan trọng: Hãy trình bày mọi thứ thật mạch lạc, dùng gạch đầu dòng, ngôn ngữ đơn giản và tập trung vào những từ khóa quan trọng nhất giúp tôi ghi nhớ nhanh để chuẩn bị cho kỳ thi.

Viết lại prompt trên theo dạng Iterative Prompting (hỏi lại để bổ sung ví dụ cụ thể về ứng dụng thực tế):

-Bây giờ, để củng cố kiến thức, tôi muốn hiểu thật rõ cách Queue được áp dụng trong thực tế. Bạn có thể giúp tôi mô phỏng hoạt động của nó trong một vài ví dụ cụ thể được không?

-Với mỗi tình huống dưới đây, vui lòng giải thích theo cấu trúc sau:

1. Tình huống ứng dụng: [Tên ứng dụng]

2. Phần tử trong hàng đợi là gì?: [Mô tả ngắn gọn, ví dụ: một tiến trình, một đỉnh đồ thị, một file tài liệu]

3. Mô phỏng từng bước:

Enqueue (Thêm vào): Giải thích sự kiện nào khiến một phần tử được thêm vào cuối hàng đợi.

Dequeue (Lấy ra): Giải thích sự kiện nào khiến phần tử ở đầu hàng đợi được lấy ra và xử lý.

-Cụ thể, tôi cần bạn giải thích cho 3 trường hợp:

Lập lịch CPU trong hệ điều hành.

Thuật toán tìm kiếm theo chiều rộng (BFS).

Hệ thống quản lý lệnh in (Print Spooling).

-Cách giải thích trực quan, từng bước như thế này sẽ giúp tôi xây dựng một "bức tranh" trong đầu về cách Queue hoạt động, thay vì chỉ học thuộc lý thuyết suông.

Viết một prompt Chain-of-thought để AI phân tích: “AI có thể đóng vai trò như thế nào trong việc phát triển các hệ thống phần mềm lớn, phức tạp trong tương lai?”:

Hãy đóng vai một nhà chiến lược công nghệ và kiến trúc sư phần mềm hàng đầu. Tôi muốn bạn phân tích sâu về câu hỏi: “AI có thể đóng vai trò như thế nào trong việc phát triển các hệ thống phần mềm lớn, phức tạp trong tương lai?”

Để có câu trả lời toàn diện, hãy sử dụng phương pháp tư duy chuỗi (Chain-of-Thought) và trình bày theo từng bước logic sau. Hãy trình bày rõ ràng suy luận của bạn ở mỗi bước.

Bước 1: Phân tích bối cảnh và thách thức hiện tại.

Đầu tiên, hãy xác định những thách thức lớn nhất trong việc phát triển các hệ thống phần mềm quy mô lớn và phức tạp hiện nay là gì? (Ví dụ: quản lý độ phức tạp, đảm bảo chất lượng, thời gian đưa sản phẩm ra thị trường, bảo trì, chi phí nhân sự,...). Việc này sẽ tạo tiền đề để xem AI có thể giải quyết những vấn đề gì.

Bước 2: Phân tích vai trò của AI theo từng giai đoạn trong Vòng đời Phát triển Phần mềm (SDLC).

Tiếp theo, hãy đi qua từng giai đoạn của SDLC và suy luận xem AI có thể được tích hợp vào đâu và mang lại giá trị gì. Với mỗi giai đoạn, hãy giải thích cụ thể:

Giai đoạn Lập kế hoạch và Phân tích Yêu cầu: AI có thể giúp tự động hóa việc thu thập yêu cầu từ người dùng, phân tích phản hồi, hoặc dự đoán mức độ ưu tiên của các tính năng không?

Giai đoạn Thiết kế và Kiến trúc: AI có thể đề xuất các mẫu thiết kế (design patterns), kiến trúc hệ thống tối ưu dựa trên yêu cầu, hoặc tự động phát hiện các điểm yếu tiềm ẩn trong thiết kế không?

Giai đoạn Lập trình (Implementation): Vai trò của AI không chỉ dừng ở việc gợi ý code (như Copilot). Nó có thể tự động viết các module hoàn chỉnh, tái cấu trúc (refactor) code, hoặc dịch mã nguồn giữa các ngôn ngữ lập trình không?

Giai đoạn Kiểm thử (Testing): AI có thể tự động tạo ra các bộ test case (kể cả các trường hợp biên), thực hiện kiểm thử tự động một cách thông minh hơn, hay thậm chí dự đoán các khu vực code dễ phát sinh lỗi nhất không?

Giai đoạn Triển khai và Vận hành (Deployment & DevOps): AI có thể tối ưu hóa quy trình CI/CD, tự động hóa việc giám sát (monitoring) và dự báo sự cố hệ thống trước khi chúng xảy ra không?

Giai đoạn Bảo trì (Maintenance): AI có thể tự động phân tích log để tìm ra nguyên nhân lỗi, hoặc thậm chí đề xuất và áp dụng các bản vá lỗi tự động không?

Bước 3: Tổng hợp và hình dung về vai trò tương lai của AI.

Dựa trên những phân tích ở Bước 2, hãy tổng hợp lại. AI sẽ chỉ là một công cụ hỗ trợ (assistant) hay sẽ tiến hóa thành một "đồng nghiệp ảo" (AI agent/teammate) có khả năng tự chủ? Hay thậm chí trở thành một "Kiến trúc sư AI" có thể quản lý toàn bộ dự án?

Bước 4: Đánh giá các thách thức và rủi ro đi kèm.

Cuối cùng, hãy suy nghĩ về những rủi ro và thách thức khi phụ thuộc nhiều vào AI trong phát triển phần mềm. (Ví dụ: vấn đề bảo mật của code do AI tạo ra, sự suy giảm kỹ năng của lập trình viên, thiên vị (bias) trong các quyết định của AI, và ai sẽ chịu trách nhiệm khi AI gây ra lỗi?).

Yêu cầu cuối cùng: Hãy trình bày câu trả lời của bạn theo đúng 4 bước trên, đảm bảo logic của bước trước là nền tảng cho bước sau để tạo thành một chuỗi lập luận chặt chẽ.

Viết prompt yêu cầu AI giải thích nguyên lý hoạt động của Binary Search Tree cho sinh viên mới học:

Hãy đóng vai một gia sư lập trình tận tâm, đang giải thích khái niệm Cây Tìm Kiếm Nhị Phân (Binary Search Tree - BST) cho một bạn sinh viên vừa mới bắt đầu học môn Cấu trúc dữ liệu.

Mục tiêu của tôi là hiểu được nguyên lý cốt lõi và cách hoạt động của nó, chứ không chỉ là định nghĩa suông. Vui lòng giải thích theo cấu trúc sau để tôi dễ theo dõi nhất:

1. Bắt đầu bằng một ví dụ đời thực:

Hãy dùng một ví dụ đơn giản và quen thuộc để giải thích ý tưởng cơ bản đằng sau BST. Ví dụ như cách bạn tìm một từ trong từ điển, hoặc cách trò chơi "đoán số" hoạt động (khi bạn luôn hỏi "lớn hơn hay nhỏ hơn?").

2. Giải thích "Nguyên Tắc Vàng" của BST:

Làm rõ quy tắc bất di bất dịch của BST một cách đơn giản nhất:

Mỗi nút (node) có một giá trị.

Tất cả các giá trị ở cây con bên trái của một nút đều nhỏ hơn giá trị của nút đó.

Tất cả các giá trị ở cây con bên phải của một nút đều lớn hơn giá trị của nút đó.

Hãy nhấn mạnh đây là quy tắc quan trọng nhất.

3. Mô phỏng hoạt động với ví dụ cụ thể:

Đây là phần quan trọng nhất. Hãy sử dụng một dãy số cụ thể để xây dựng một cây BST từ đầu và minh họa các thao tác chính.

Dãy số để xây dựng cây: ``

Thao tác Chèn (Insert): Hãy giải thích từng bước cách bạn chèn số `65` vào cây đã có. Bắt đầu từ gốc (50), so sánh và đi sang trái hay phải cho đến khi tìm được vị trí trống.

Thao tác Tìm kiếm (Search): Sau đó, hãy giải thích từng bước cách bạn tìm số `40` trong cây. Mô tả quá trình so sánh và di chuyển cho đến khi tìm thấy giá trị.

4. Tại sao chúng ta cần BST?

Cuối cùng, hãy giải thích ngắn gọn lợi ích lớn nhất của việc sử dụng BST so với việc lưu dữ liệu trong một mảng hoặc danh sách thông thường. Hãy đề cập đến tốc độ tìm kiếm, chèn, xóa (nói về ý tưởng "chia đôi phạm vi tìm kiếm mỗi lần" thay vì phải duyệt qua toàn bộ).

Yêu cầu: Vui lòng sử dụng ngôn ngữ thân thiện, đơn giản, ví như bạn đang nói chuyện trực tiếp với tôi. Hãy dùng gạch đầu dòng để trình bày các bước cho rõ ràng. Mục tiêu cuối cùng là giúp một người chưa biết gì có thể hình dung và hiểu được cách BST hoạt động.

Viết 1 prompt chi tiết yêu cầu AI sinh ví dụ code minh họa một hàm tìm số lớn nhất trong mảng, sau đó yêu cầu AI giải thích cùng một khái niệm ở 3 cấp độ khác nhau: cho người mới học, cho sinh viên CNTT, và cho kỹ sư phần mềm.

Hãy đóng vai một chuyên gia lập trình và người hướng dẫn kỹ thuật dày dạn kinh nghiệm. Tôi muốn bạn thực hiện một yêu cầu gồm hai phần: đầu tiên là viết một hàm đơn giản, và sau đó là giải thích nó ở ba cấp độ chuyên môn khác nhau để phục vụ cho các mục đích học tập khác nhau.

Phần 1: Viết Code Minh Họa

Hãy viết một hàm bằng ngôn ngữ lập trình Python có tên là `find\_max`.

Chức năng: Hàm này nhận đầu vào là một danh sách (list) các số nguyên và trả về số nguyên lớn nhất trong danh sách đó.

Xử lý trường hợp đặc biệt: Vui lòng xử lý trường hợp danh sách đầu vào bị rỗng (empty list) bằng cách trả về `None`.

Định dạng: Hãy đặt code trong khối markdown để tôi dễ dàng sao chép.

Phần 2: Giải Thích Theo 3 Cấp Độ

Sau khi cung cấp code, hãy giải thích chính xác cách hoạt động của hàm đó cho ba đối tượng khác nhau. Với mỗi đối tượng, hãy điều chỉnh ngôn ngữ, mức độ chi tiết kỹ thuật và các khái niệm liên quan cho phù hợp.

Cấp độ 1: Giải thích cho người mới học lập trình (Beginner)

Đối tượng: Một người chưa từng lập trình, hoặc chỉ mới biết những khái niệm cơ bản nhất.

Yêu cầu:

Sử dụng một ví dụ đời thực cực kỳ đơn giản để minh họa (ví dụ: tìm người cao nhất trong một lớp học).

Giải thích từng dòng code như một câu lệnh bằng ngôn ngữ tự nhiên. Ví dụ: "Dòng này tạo một cái hộp tên là 'max\_so\_far' và tạm thời đặt số đầu tiên vào đó...".

Tránh hoàn toàn các thuật ngữ kỹ thuật phức tạp. Tập trung vào logic "lặp qua từng số và so sánh".

Mục tiêu là giúp họ hình dung được luồng suy nghĩ của máy tính.

Cấp độ 2: Giải thích cho sinh viên Công nghệ thông tin (CS Student)

Đối tượng: Một sinh viên năm nhất hoặc năm hai đã học các khóa nhập môn về lập trình và cấu trúc dữ liệu.

Yêu cầu:

Sử dụng các thuật ngữ kỹ thuật chính xác: khởi tạo (initialization), vòng lặp (iteration), biến tạm (temporary variable), so sánh (comparison), câu lệnh điều kiện (conditional statement).

Phân tích độ phức tạp thời gian (Time Complexity) của thuật toán. Giải thích tại sao nó là $$O(n)$$ và ý nghĩa của việc này đối với hiệu năng khi danh sách lớn lên.

Nhắc đến các trường hợp biên (edge cases) mà hàm đã xử lý (mảng rỗng) và các trường hợp khác cần lưu ý (mảng có một phần tử, mảng có các số trùng nhau).

Mục tiêu là kết nối đoạn code này với các kiến thức lý thuyết về thuật toán.

Cấp độ 3: Giải thích cho Kỹ sư Phần mềm có kinh nghiệm (Software Engineer)

Đối tượng: Một lập trình viên chuyên nghiệp đang tham gia vào một buổi đánh giá code (code review).

Yêu cầu:

Tập trung vào tính ngắn gọn, dễ đọc (conciseness, readability) và các phương pháp tốt nhất (best practices).

Đề xuất một cách viết "Pythonic" hơn, tức là cách viết tự nhiên và hiệu quả hơn trong Python, bằng cách sử dụng các hàm tích hợp sẵn (built-in function) như `max()`.

Phân tích ưu và nhược điểm của việc tự triển khai vòng lặp so với việc dùng hàm có sẵn (ví dụ: tính rõ ràng, hiệu suất, khả năng bảo trì).

Bàn luận về các vấn đề có thể xảy ra trong môi trường sản xuất (production), ví dụ: đầu vào không phải là danh sách số nguyên mà là danh sách chứa nhiều kiểu dữ liệu khác nhau, và cách xử lý chúng một cách an toàn.

Ngôn ngữ nên súc tích, chuyên nghiệp và đi thẳng vào vấn đề.

Viết 1 prompt yêu cầu AI sinh code một hàm Python kiểm tra một số có phải là số nguyên tố (đầu vào n).

Hãy đóng vai một lập trình viên Python kinh nghiệm. Nhiệm vụ của bạn là viết một hàm hoàn chỉnh và được tối ưu hóa để kiểm tra xem một số nguyên có phải là số nguyên tố hay không.

Yêu cầu chi tiết:

1. Tên hàm: `is\_prime`

2. Tham số đầu vào: Một số nguyên `n`.

3. Giá trị trả về:

Trả về `True` nếu `n` là một số nguyên tố.

Trả về `False` nếu `n` không phải là một số nguyên tố.

Logic và các trường hợp cần xử lý:

Định nghĩa số nguyên tố: Một số nguyên tố là một số tự nhiên lớn hơn 1 và không có ước số dương nào khác ngoài 1 và chính nó.

Xử lý các trường hợp biên (Edge Cases): Hàm của bạn phải xử lý chính xác các trường hợp `n` nhỏ hơn hoặc bằng 1. Theo định nghĩa, các số này không phải là số nguyên tố.

Tối ưu hóa thuật toán:

Không cần phải kiểm tra tất cả các số từ 2 đến `n-1`.

Bạn chỉ cần kiểm tra các ước số từ 2 cho đến căn bậc hai của `n`. Nếu không tìm thấy ước nào trong khoảng này, `n` chính là số nguyên tố.

Bạn có thể thêm một bước kiểm tra nhanh: nếu `n` lớn hơn 2 và là số chẵn, nó chắc chắn không phải là số nguyên tố.

Yêu cầu về chất lượng Code:

Docstring: Vui lòng viết một docstring rõ ràng ngay bên dưới dòng định nghĩa hàm, giải thích chức năng của hàm, tham số đầu vào (`@param`) và giá trị trả về (`@return`).

Chú thích (Comments): Thêm chú thích ngắn gọn cho các phần logic quan trọng, đặc biệt là phần xử lý trường hợp biên và phần tối ưu hóa vòng lặp.

Định dạng: Đặt toàn bộ code Python vào trong một khối markdown.

Ví dụ kết quả mong đợi:

`is\_prime(7)` sẽ trả về `True`.

`is\_prime(10)` sẽ trả về `False`.

`is\_prime(1)` sẽ trả về `False`.

`is\_prime(2)` sẽ trả về `True`.

Tình huống: Bạn cần chuẩn bị bài thuyết trình 10 phút cho buổi Workshop Thực tiễn phát triển phần mềm Agile với chủ đề: “AI có thể tối ưu hóa quy trình Agile như thế nào?”:

Tạo outline 3 phần (Mở đầu – Nội dung – Kết luận)

Gợi ý 5 slide với nội dung gạch đầu dòng

Đưa ra 2 ví dụ thực tế về cách AI hỗ trợ các hoạt động Agile (VD: Ước lượng task, tạo user story,...)

Viết script ngắn (~2 phút) để luyện nói phần mở đầu, giới thiệu tổng quan về vai trò của AI trong Agile

Phần 1: Mở đầu (1.5 phút)

Giới thiệu: Đặt vấn đề với những thách thức thường gặp trong quy trình Agile (ví dụ: ước lượng không chính xác, quản lý backlog tốn thời gian).

Giới thiệu chủ đề: AI như một "trợ lý thông minh" có thể giúp giải quyết các thách thức đó.

Mục tiêu bài nói: Khám phá các ứng dụng cụ thể của AI để nâng cao hiệu quả, không phải để thay thế con người.

Phần 2: Nội dung chính (7 phút)

AI trong giai đoạn Lập kế hoạch (Planning):

Tự động tạo và phân tích User Story từ các yêu cầu cấp cao.

Sử dụng AI để ước lượng (estimate) task dựa trên dữ liệu lịch sử, giảm thiểu sự chủ quan.

Hỗ trợ sắp xếp độ ưu tiên cho backlog (prioritization) dựa trên giá trị kinh doanh và mức độ rủi ro.

AI trong giai đoạn Phát triển & Kiểm thử (Development & Testing):

Các công cụ AI hỗ trợ viết code (code assistant) và review code tự động.

Tạo test case thông minh, xác định các vùng code có rủi ro cao cần kiểm thử kỹ.

Phân tích và dự báo bug tiềm ẩn.

AI trong giai đoạn Vận hành & Cải tiến (Operations & Retrospective):

Dự báo rủi ro của Sprint (ví dụ: khả năng không hoàn thành mục tiêu).

Phân tích dữ liệu từ các buổi retrospective để tìm ra các vấn đề cốt lõi của team.

Phần 3: Kết luận (1.5 phút)

Tóm tắt lợi ích chính: Tăng năng suất, cải thiện chất lượng sản phẩm, và đưa ra quyết định dựa trên dữ liệu.

Nhấn mạnh thông điệp: AI là công cụ hỗ trợ đắc lực, giúp đội ngũ Agile tập trung vào sự sáng tạo và giải quyết vấn đề phức tạp.

Kêu gọi hành động: Khuyến khích khán giả bắt đầu tìm hiểu và thử nghiệm các công cụ AI nhỏ trong dự án của mình.

Gợi ý nội dung cho 5 slide

Slide 1: Tiêu đề

AI CÓ THỂ TỐI ƯU HÓA QUY TRÌNH AGILE NHƯ THẾ NÀO?

Tên của bạn

Workshop Thực tiễn phát triển phần mềm Agile

Slide 2: Những thách thức trong Agile & Vai trò của AI

Thách thức thường gặp:

Ước lượng (estimation) tốn thời gian và thiếu chính xác.

Quản lý backlog phức tạp.

Phát hiện lỗi muộn trong quy trình.

Khó đo lường và dự báo rủi ro.

AI - Người trợ lý thông minh:

AI không thay thế, AI hỗ trợ và tăng cường năng lực cho đội nhóm.

Slide 3: AI trong Lập kế hoạch & Ước lượng

Tạo User Story tự động: AI phân tích yêu cầu thô và tạo ra các user story có cấu trúc chuẩn.

Ước lượng thông minh:

Phân tích các task đã hoàn thành (mô tả, độ phức tạp, người thực hiện).

Đưa ra gợi ý về story point hoặc thởi gian thực hiện cho task mới.

Sắp xếp Backlog thông minh: AI gợi ý độ ưu tiên dựa trên giá trị, rủi ro và sự phụ thuộc.

Slide 4: AI trong Phát triển & Kiểm thử

Code Assistant (Trợ lý lập trình):

Gợi ý code, tìm lỗi cú pháp, và đề xuất giải pháp tối ưu (VD: GitHub Copilot).

Kiểm thử thông minh (AI-powered Testing):

Tự động tạo bộ test case dựa trên user story.

Xác định những phần code thay đổi có rủi ro cao để tập trung kiểm thử.

Slide 5: Kết luận & Hướng đi tương lai

Lợi ích chính:

Nhanh hơn: Tự động hóa các tác vụ lặp lại.

Chính xác hơn: Quyết định dựa trên dữ liệu thay vì cảm tính.

Thông minh hơn: Giải phóng con người để tập trung vào sáng tạo.

Tương lai: AI sẽ ngày càng tích hợp sâu hơn vào các công cụ ALM (Application Lifecycle Management) như Jira, Trello.

Cảm ơn & Câu hỏi?

2 ví dụ thực tế về cách AI hỗ trợ Agile

Ví dụ 1: Ước lượng Task (Task Estimation)

Tình huống: Một team Agile chuẩn bị cho buổi Sprint Planning và cần ước lượng story point cho một loạt user story mới. Quá trình này thường gây tranh cãi và thiếu nhất quán.

Cách AI hỗ trợ: Team sử dụng một công cụ AI được tích hợp vào Jira. Công cụ này đã được "học" từ hàng ngàn task đã hoàn thành trong lịch sử của công ty. Khi xem một user story mới, AI sẽ phân tích mô tả, độ phức tạp của yêu cầu, và so sánh với các task tương tự trong quá khứ. Sau đó, nó sẽ đưa ra một gợi ý, ví dụ: "Dựa trên dữ liệu, task này có 75% khả năng là 5 story points và 25% khả năng là 8 story points." Gợi ý này trở thành một điểm tham chiếu khách quan, giúp team thảo luận và đưa ra quyết định nhanh chóng và nhất quán hơn.

Ví dụ 2: Tạo User Story và Acceptance Criteria

Tình huống: Product Owner (PO) có một ý tưởng về tính năng mới: "Tôi muốn người dùng có thể tìm kiếm sản phẩm bằng giọng nói trên ứng dụng di động." Tuy nhiên, PO không có nhiều thởi gian để viết chi tiết các user story và tiêu chí chấp nhận (acceptance criteria).

Cách AI hỗ trợ: PO nhập yêu cầu cấp cao đó vào một công cụ AI tạo sinh. AI sẽ ngay lập tức trả về kết quả:

User Story: "Là một người dùng ứng dụng di động, tôi muốn tìm kiếm sản phẩm bằng giọng nói để có thể tìm hàng nhanh hơn mà không cần gõ phím."

Gợi ý Acceptance Criteria:

1. Nút tìm kiếm bằng giọng nói phải hiển thị rõ ràng trên thanh tìm kiếm.

2. Khi người dùng nhấn nút, ứng dụng phải yêu cầu quyền truy cập micro.

3. Hệ thống phải chuyển đổi chính xác giọng nói thành văn bản và thực hiện tìm kiếm.

4. Nếu không nhận diện được giọng nói, hệ thống phải hiển thị thông báo lỗi thân thiện.

PO chỉ cần rà soát, chỉnh sửa và bổ sung, tiết kiệm được rất nhiều thởi gian so với việc viết từ đầu.

Script luyện nói phần mở đầu (~2 phút)

"Xin chào tất cả mọi người!

Tôi muốn bắt đầu bằng một câu hỏi: Đã bao giờ trong một buổi Sprint Planning, cả team của bạn tranh cãi hàng giờ chỉ để ước lượng một vài task chưa? Hay đã bao giờ bạn cảm thấy kiệt sức vì phải viết đi viết lại hàng chục user story với những tiêu chí gần như giống hệt nhau?

Đây là những thách thức rất quen thuộc trong thế giới Agile. Chúng ta theo đuổi sự linh hoạt và hiệu quả, nhưng đôi khi lại bị sa lầy vào những công việc lặp đi lặp lại và những quyết định cảm tính.

Nhưng hôm nay, tôi muốn giới thiệu một 'người đồng đội' mới có thể giúp chúng ta giải quyết những vấn đề này. Đó chính là Trí tuệ nhân tạo, hay AI.

Khi nghe đến AI trong phát triển phần mềm, nhiều người có thể lo lắng về việc bị thay thế. Nhưng tôi muốn các bạn hãy nhìn nhận AI theo một cách khác: AI không phải là đối thủ, mà là một trợ lý thông minh. Vai trò của nó không phải là để thay thế lập trình viên hay Product Owner. Vai trò của nó là giúp chúng ta tự động hóa các công việc nhàm chán, cung cấp những phân tích sâu sắc dựa trên dữ liệu, và quan trọng nhất là giải phóng bộ não của chúng ta để tập trung vào những việc thực sự cần đến sự sáng tạo và tư duy chiến lược.

Trong 10 phút tới, chúng ta sẽ cùng nhau khám phá xem người trợ lý AI này thực sự có thể tối ưu hóa quy trình Agile của chúng ta như thế nào, từ việc lập kế hoạch, ước lượng, cho đến giai đoạn phát triển và kiểm thử.

Hãy cùng bắt đầu nhé!"

Viết 1 prompt yêu cầu AI phân tích và sửa lỗi trong đoạn code sau (được lưu trữ trong file string\_ops.py):

Hãy đóng vai một chuyên gia lập trình Python có kinh nghiệm. Tôi có một đoạn code Python đang gặp lỗi và cần bạn giúp phân tích cũng như đưa ra giải pháp sửa chữa.

Đoạn code có vấn đề:

```python

def concatenate\_strings(str1, str2):

result = str1 + str3

return result

print(concatenate\_strings("Hello", "World"))

```

Yêu cầu cụ thể:

1. Phân tích lỗi:

- Xác định chính xác lỗi gì đang xảy ra trong code trên.

- Giải thích tại sao lỗi này lại xảy ra (nguyên nhân gốc rễ).

- Dự đoán thông báo lỗi mà Python sẽ hiển thị khi chạy code này.

2. Đưa ra giải pháp sửa chữa:

- Sửa lỗi trong code và cung cấp phiên bản đã được sửa chữa.

- Giải thích từng thay đổi bạn đã thực hiện.

- Đảm bảo code sau khi sửa sẽ chạy đúng và cho ra kết quả mong đợi.

3. Kiểm tra và cải tiến:

- Chạy thử code đã sửa với ví dụ đã cho: `concatenate\_strings("Hello", "World")`.

- Dự đoán kết quả đầu ra sau khi sửa.

- Đề xuất thêm một vài cải tiến nhỏ để làm cho hàm này robust hơn (ví dụ: xử lý trường hợp đầu vào không phải là string).

4. Bài học rút ra:

- Đưa ra 2-3 lời khuyên để tránh loại lỗi tương tự trong tương lai.

- Giải thích tầm quan trọng của việc đặt tên biến chính xác và nhất quán.

Yêu cầu về format: Vui lòng trình bày câu trả lời theo đúng 4 phần trên, sử dụng gạch đầu dòng và code block markdown để dễ đọc.

Viết 2 prompt mẫu sẽ nhập trong Cursor để:

Prompt 1: Yêu cầu Cursor sinh unit tests cho một hàm Python (ví dụ: hàm add(a, b))

Hãy tạo một bộ unit tests hoàn chỉnh cho hàm `add(a, b)` trong file này. Yêu cầu cụ thể:

1. Sử dụng framework pytest để viết các test cases

2. Tạo các test cases bao gồm:

- Test các trường hợp bình thường (positive cases): số nguyên dương, số thực, số âm

- Test các trường hợp biên (edge cases): số 0, số rất lớn, số rất nhỏ

- Test các trường hợp ngoại lệ (negative cases): đầu vào không phải số (string, None, list, dict)

- Test performance với số lượng lớn phép tính

3. Mỗi test case phải có:

- Tên test function mô tả rõ ràng (test\_add\_positive\_numbers, test\_add\_with\_zero, etc.)

- Docstring giải thích mục đích của test

- Sử dụng pytest.parametrize cho các test cases tương tự

- Assert statements với thông báo lỗi rõ ràng

4. Thêm fixtures nếu cần thiết cho setup/teardown

5. Đảm bảo test coverage đạt 100% cho hàm add()

6. Tạo file test riêng biệt với tên test\_[tên\_file\_gốc].py

7. Thêm các import statements cần thiết

Hãy tạo code test chất lượng cao, tuân thủ best practices của pytest và Python testing.

A screenshot of a computer

AI-generated content may be incorrect.

A black screen with a black background

AI-generated content may be incorrect.

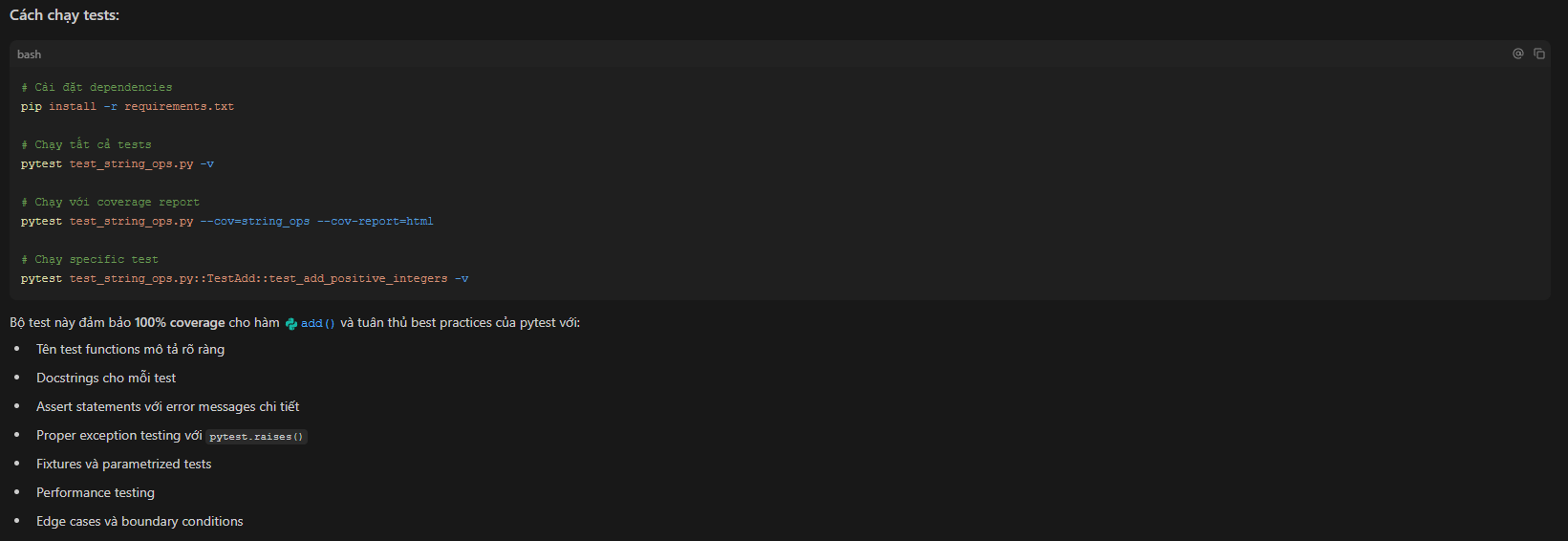
A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a person

AI-generated content may be incorrect.





Prompt 2: Yêu cầu Cursor sửa đổi code toàn cục để chuyển if/else lồng nhau thành match/case

Hãy phân tích toàn bộ project này và tự động chuyển đổi tất cả các câu lệnh if/else lồng nhau phức tạp thành cú pháp match/case của Python 3.10+. Yêu cầu cụ thể:

1. Quét tất cả các file .py trong project để tìm:

- Các chuỗi if/elif/else có từ 3 điều kiện trở lên

- Các nested if statements (if lồng nhau) có độ sâu >= 2 levels

- Các pattern matching phù hợp để chuyển sang match/case

2. Chỉ thực hiện conversion khi:

- Logic có thể được cải thiện về mặt readability

- Không làm thay đổi behavior của code

- Phù hợp với pattern matching syntax của Python

3. Đối với mỗi conversion:

- Giữ nguyên logic và kết quả của code gốc

- Thêm comment giải thích sự thay đổi

- Sử dụng các pattern phù hợp: literal patterns, capture patterns, wildcard patterns, guard clauses

- Xử lý các trường hợp đặc biệt như multiple conditions, complex expressions

4. Tạo backup của các file gốc trước khi modify

5. Tạo summary report về:

- Số lượng file được modify

- Số lượng if/else blocks được convert

- Các trường hợp không thể convert và lý do

- Estimated improvement về code readability

6. Đảm bảo compatibility:

- Kiểm tra Python version requirement (>= 3.10)

- Thêm version check nếu cần thiết

- Maintain backward compatibility nếu có yêu cầu

Lưu ý: Chỉ thực hiện conversion khi chắc chắn 100% về tính đúng đắn. Nếu có nghi ngờ, hãy đề xuất thay vì tự động thay đổi.

