DUMPSBOSS.

GitHub Copilot Exam

Microsoft GH-300

Version Demo

Total Demo Questions: 19

Total Premium Questions: 198

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, Responsible AI	31
Topic 2, GitHub Copilot plans and features	72
Topic 3, How GitHub Copilot works and handles data	33
Topic 4, Prompt engineering and prompt crafting	19
Topic 5, Developer use cases for AI	27
Topic 6, Testing with GitHub Copilot	14
Topic 7, Mix Questions	2
Total	198

QUESTION NO: 1

How can GitHub Copilot Chat help verify that a function behaves correctly?

A. By suggesting assertions based on the code's context and semantics.

B. By automatically writing all the tests for the function.

C. By writing the implementation code for the function.

D. By executing the test cases to validate the correctness of the code.

ANSWER: A

Explanation:

By suggesting assertions based on the code's context and semantics is correct because assertions convert expected behavior into concrete checks that can be evaluated by a test framework. When a developer asks GitHub Copilot Chat to generate or improve tests, Copilot can consider available context such as the function signature, parameter and variable names, comments, calling code, and implementation details. From that context, it can propose assertions for expected return values, state changes, exceptions, and representative edge cases. These assertions give the developer a practical starting point for expressing what the function must do for particular inputs.

Assertions are central to unit testing because they compare an observed result with an expected result and make the function's behavioral contract explicit. Copilot Chat can assist in drafting these test checks and in explaining or refining them, while the developer reviews the suggestions, selects meaningful cases, and runs the tests in the project environment. GitHub documents Copilot Chat's support for code-generation and testing-related tasks in its Copilot Chat documentation. For guidance on creating focused, reliable unit tests and assertions, see Microsoft's unit testing best practices.

QUESTION NO: 2

In Visual Studio, what is the primary purpose of the `/optimize` slash command in GitHub Copilot Chat?

A. Translates code into a more performant language.

B. Automatically formats the code according to the selected style guide.

C. Summarizes your documentation into more maintainable and readable formats.

D. Analyzes the selected code and suggests changes to improve its performance.

ANSWER: D

Explanation:

`Analyzes the selected code and suggests changes to improve its performance.` is correct because the `/optimize` command directs GitHub Copilot Chat to examine the currently selected code with a focus on opportunities to make it more efficient. Copilot can identify potential performance improvements and propose code changes that may reduce unnecessary work, improve data handling, or otherwise make the implementation more performant. This command is intended to help developers reason about optimization within the context of their code rather than performing a compiler-style translation or a formatting operation. The developer remains responsible for reviewing the proposed changes, validating that they preserve required behavior, and measuring their impact through appropriate profiling, testing, and benchmarking. GitHub Copilot Chat supports slash commands as concise instructions that tailor the kind of assistance Copilot provides in an IDE. For more information about using Copilot Chat prompts and commands in an IDE, see GitHub Docs: Asking GitHub Copilot questions in your IDE and Microsoft Learn: GitHub Copilot Chat in Visual Studio.

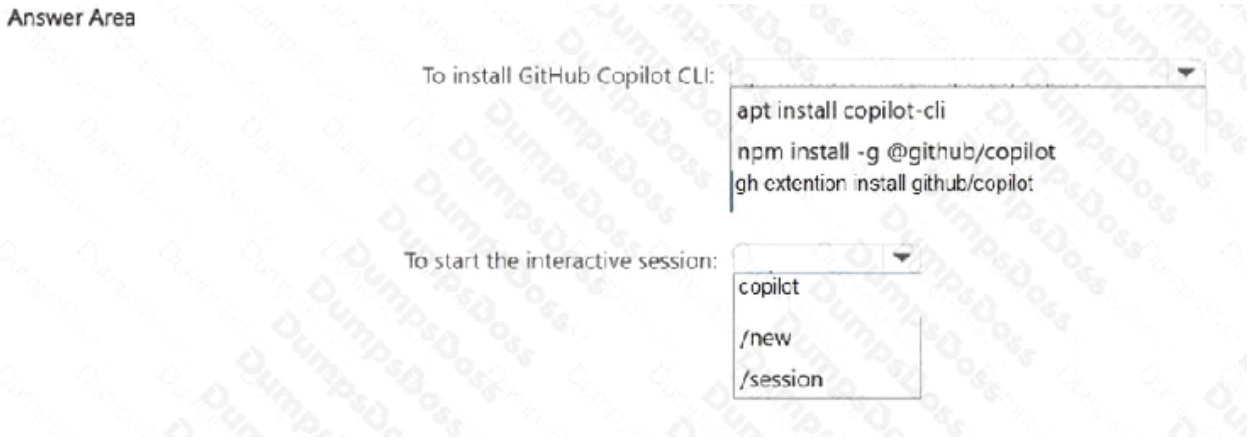
QUESTION NO: 3 - (HOTSPOT)

You are deploying a new development environment to a Windows 11 device.

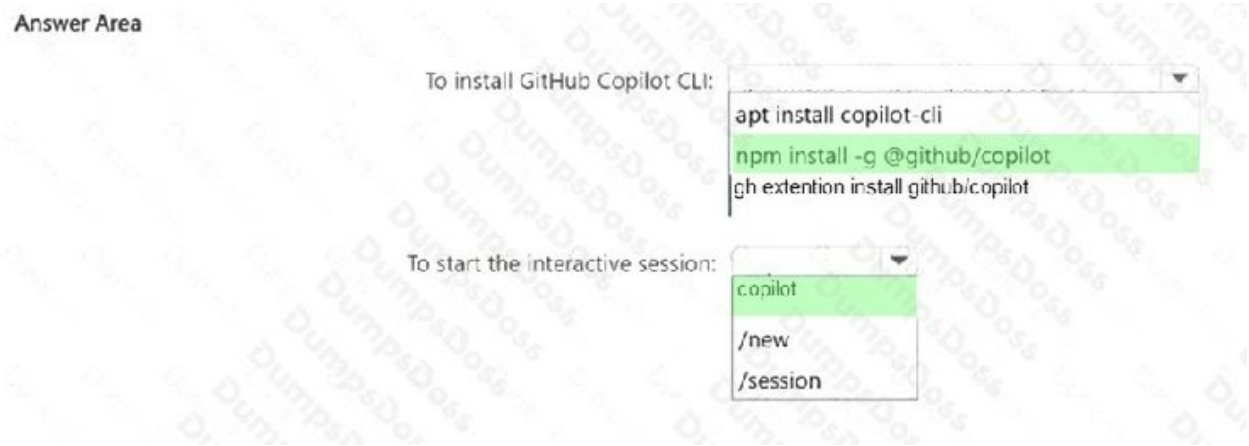
You need to install GitHub Copilot CLI, and then start an interactive GitHub Copilot CLI session.

Which command should you run for each requirement? To answer, select the appropriate options in the answer area.

NOTE: Each correct selection is worth one point.



ANSWER:



Explanation:

GitHub Copilot CLI is installed through npm as the `@github/copilot` package. Using `npm install -g @github/copilot` installs it globally, which makes the `copilot` command available from terminals across the Windows device rather than limiting it to one specific project directory. This is the appropriate approach for preparing a general development environment, provided Node.js and npm are installed.

After the global package installation finishes, entering `copilot` at a terminal prompt starts the GitHub Copilot CLI interactive experience. The interactive session lets the developer submit natural-language requests and work with Copilot directly from the command line. This is the normal invocation command for the standalone CLI and is separate from installing a GitHub CLI extension.

The selected commands correctly implement the required sequence: install the globally available GitHub Copilot CLI package first, then invoke the CLI executable to begin the interactive session. GitHub's current installation documentation lists npm installation for Copilot CLI, and its CLI documentation describes invoking the tool through the `copilot` command. See Install GitHub Copilot CLI and About GitHub Copilot CLI.

QUESTION NO: 4

Which two actions can improve the context GitHub Copilot uses to generate a response or suggestion? Each correct answer contributes to the solution.

A. By opening the relevant tabs in your IDE

B. By adding relevant code snippets to your prompt

C. By adding the important files to your .gitconfig

D. By adding the full file paths to your prompt of important files

ANSWER: A B

Explanation:

By opening the relevant tabs in your IDE and by adding relevant code snippets to your prompt, you give GitHub Copilot useful information about the task and codebase. Copilot can use contextual information available in the editor, such as the active file, open files, selected code, surrounding code, symbols, and project patterns. Opening source files that are directly related to the current change can therefore help Copilot align a suggestion with existing types, method signatures, naming conventions, and implementation approaches.

Adding relevant code snippets to a prompt provides direct and explicit context, particularly when you need help with a specific function, error, interface, test case, or expected behavior. A focused snippet lets you show the actual inputs, dependencies, and desired result rather than requiring Copilot to infer them. GitHub recommends giving Copilot relevant context and clearly describing the goal; in IDE chat, users can also select code or reference files and symbols to ground a request. These practices improve the relevance and practical usefulness of Copilot responses. See Asking GitHub Copilot questions in your IDE and Using GitHub Copilot Chat in your IDE.

QUESTION NO: 5

After discussing a critical API compatibility requirement in one GitHub Copilot Chat conversation, a developer starts a new conversation. Copilot's first response in the new conversation does not account for that requirement. What should the developer do next?

A. Restate the compatibility requirement and include the relevant API details in the new conversation.

B. Assume that all requirements from the earlier conversation will be applied automatically.

C. Accept the response because API compatibility can be addressed after deployment.

D. Open unrelated repository files to provide additional context.

ANSWER: A

Explanation:

Restate the compatibility requirement and include the relevant API details in the new conversation. GitHub Copilot Chat uses the information available in the current conversation and the context explicitly supplied to it, such as a focused prompt, selected code, referenced files, or relevant documentation. Starting a new conversation is useful for changing topics or resetting the discussion, but it also means the developer should provide essential constraints again rather than relying on prior discussion to guide the new response.

For an API compatibility task, the restated context should be specific enough to make the expected behavior actionable. For example, the developer can identify the supported API version, describe required backward-compatible request or response shapes, mention deprecated fields that must remain supported, and provide the relevant interface or endpoint details. This gives Copilot a clear, grounded basis for generating code or recommendations that honor the requirement. GitHub's guidance emphasizes providing relevant context and using clear, specific prompts to improve Copilot Chat responses. See Asking GitHub Copilot questions in your IDE and Prompt engineering for GitHub Copilot.

QUESTION NO: 6

Which GitHub Copilot configuration should be enabled to help reduce the risk of intellectual property infringement?

A. Blocking public code matches

B. Blocking license check configuration

C. Allowing public code matches

D. Allowing license check configuration

ANSWER: A

Explanation:

Blocking public code matches is the appropriate configuration because GitHub Copilot can detect when certain code-completion suggestions closely match public code on GitHub. When an organization enables the policy to block those matches, Copilot suppresses matching suggestions instead of presenting them to the developer. This provides a preventative control that reduces the likelihood of developers incorporating generated code that could have provenance, attribution, or license obligations associated with public source code.

For GitHub Copilot Business and GitHub Copilot Enterprise, administrators can manage this behavior as an organization policy, which ensures that the safeguard is applied consistently across covered users. The public-code matching filter is specifically intended to support organizational compliance and risk-management practices around Copilot-generated code. It should be used alongside established secure-development practices, including code review and legal or compliance review where appropriate, because developers and organizations remain responsible for evaluating code before using it in a project. See GitHub's Copilot organization policy documentation and its public code filter guidance.

QUESTION NO: 7

In the GitHub Copilot Chat data flow, what is the purpose of preprocessing a user's input?

A. It filters out irrelevant information from the user ' s input prompt.

B. It enriches the input prompt with additional context before passing it to the language model.

C. It directly generates a response based on the user ' s input prompt.

D. It formats the output response before presenting it to the user.

ANSWER: B

Explanation:

It enriches the input prompt with additional context before passing it to the language model. A developer's typed chat message alone may not contain enough information for Copilot Chat to provide a useful, codebase-aware response. Before the model request is made, Copilot can construct a richer prompt by combining the request with relevant context from the current interaction and development environment. Depending on the feature and permissions in use, this context can include selected code, the active file, open files, repository information, preceding conversation messages, and other workspace signals. This prompt construction helps the model understand not only what the developer asked, but also the technical setting in which they asked it. The language model then uses that enriched request to generate an answer, explanation, or suggested code that is more relevant to the task. GitHub documents that Copilot uses context to improve the relevance of its responses and describes how contextual information is gathered and used in Copilot features. See About GitHub Copilot context and Overview of GitHub Copilot architecture.

QUESTION NO: 8

GitHub Copilot Chat generates proposed unit tests for a function that calculates customer discounts. Before depending on these tests, which two actions should the developer perform? Select two.

A. Verify that the assertions reflect the documented business rules and boundary conditions.

B. Run the tests with the project's normal test suite.

C. Merge the tests immediately if they compile without errors.

D. Assume that accepted suggestions provide complete test coverage.

ANSWER: A B

Explanation:

Verify that the assertions reflect the documented business rules and boundary conditions. Unit tests are valuable only when their expected outcomes accurately express the application's intended behavior. For a customer-discount calculation, the developer should compare generated assertions with the documented discount criteria, including threshold values, eligible customer categories, rounding rules, and invalid or exceptional inputs. This review ensures that the tests validate the real requirements rather than merely validating assumptions inferred by the AI.

Run the tests with the project's normal test suite. Generated tests must be exercised in the repository's actual development environment to confirm that they compile, execute successfully, use the correct test framework and dependencies, and interact appropriately with the relevant code. GitHub's guidance emphasizes that developers remain responsible for reviewing and validating Copilot-generated code and should test it before use. Reviewing generated tests and executing them as part of the existing quality workflow provides practical validation that they test the intended behavior. See GitHub Copilot Chat responsible use and Using GitHub Copilot code review.

QUESTION NO: 9

Which potential legal and ethical risks should developers consider when using GitHub Copilot? Select three.

A. GitHub Copilot may accidentally suggest hard-coded secrets from its training data.

B. GitHub Copilot's suggestions may contain licensed code without providing necessary attribution.

C. GitHub Copilot may suggest code that contains security vulnerabilities.

D. GitHub Copilot may decrease developer productivity while processing far-fetched requests.

ANSWER: A B C

Explanation:

GitHub Copilot may accidentally suggest hard-coded secrets from its training data because generative coding tools can reproduce sensitive-looking strings or suggest credentials, tokens, connection strings, and other secret material in generated code. Developers must review suggestions and use secret-scanning, secure secret storage, and repository controls rather than accepting generated values as safe. GitHub documents that Copilot output should be reviewed and tested before use, and that developers remain responsible for the code they accept.

GitHub Copilot's suggestions may contain licensed code without providing necessary attribution because generated output can resemble publicly available code. This creates an intellectual-property and license-compliance consideration: organizations should assess provenance, applicable license obligations, and whether attribution or other compliance actions are required before incorporating code. GitHub provides code-referencing features and guidance to help users evaluate matches with public code.

GitHub Copilot may suggest code that contains security vulnerabilities because suggestions are generated from context and patterns rather than being a guarantee of secure, production-ready code. Secure development practices, including human review, testing, dependency checks, and security scanning, remain essential. See GitHub's responsible use guidance and GitHub's public-code matching policy documentation.

QUESTION NO: 10

Which limitations might you encounter when using GitHub Copilot Chat? Select two answers.

A. Limited training data

B. Ability to handle complex code structures

C. No biases in code suggestions

D. Limited support quality for some programming languages

ANSWER: A D

Explanation:

Limited training data is a potential limitation because GitHub Copilot Chat generates responses using an underlying AI model whose knowledge is based on data available during training, rather than complete, real-time knowledge of every codebase, framework, dependency, or recent change. Its output can therefore be incomplete, outdated, or insufficiently tailored to a particular repository unless developers provide useful context and carefully validate the result. GitHub explicitly advises users to review, test, and validate generated code and answers before relying on them.

Limited support quality for some programming languages is also a limitation. Copilot can help across many programming languages, but suggestion quality can vary with the amount of relevant code, documentation, and examples represented in the data used to develop the model. Developers working with less common, highly specialized, or newer languages and frameworks may need to supply more context and perform additional review. GitHub documents both responsible-use considerations and language support information in its Responsible use of GitHub Copilot Chat guidance and supported programming languages documentation.

QUESTION NO: 11

During AI system development, which practice helps minimize bias?

A. Collect massive amounts of data for training.

B. Focus on accuracy of the data.

C. Use diverse data, fairness metrics, and human oversight.

D. Improve computational efficiency and speed.

ANSWER: C

Explanation:

Use diverse data, fairness metrics, and human oversight. is correct because minimizing AI bias requires a deliberate, end-to-end responsible AI process rather than optimizing for a single technical characteristic. Training and evaluation data should represent the people, contexts, and scenarios the system is expected to serve. Developers should then assess outcomes using appropriate fairness measures to identify disparities across relevant groups. Human oversight is also essential: people must review the system's intended use, validate outputs, investigate potential harms, and make informed decisions about deployment and ongoing monitoring. Microsoft's responsible AI guidance identifies fairness as a core principle and describes the need to understand and mitigate unfair bias throughout an AI system's lifecycle. GitHub Copilot users and organizations can apply the same principle by reviewing generated outputs, using human judgment, and establishing governance practices appropriate to the software's impact. These combined practices help teams detect, measure, and reduce unfair outcomes before and after release. See Microsoft's responsible AI overview and Microsoft guidance for responsible AI practices.

QUESTION NO: 12

Which practices can improve the quality and relevance of suggestions generated by GitHub Copilot? Choose three.

A. Using meaningful variable names

B. Clearly defining the problem or task

C. Including personal information in the code comments

D. Providing examples of desired output

E. Use a gitignore file to exclude irrelevant files

ANSWER: A B D

Explanation:

Using meaningful variable names improves the contextual signals available to GitHub Copilot. Descriptive identifiers communicate the purpose, type, and intended use of values, helping Copilot infer appropriate logic and produce suggestions that fit the surrounding code. Clearly defining the problem or task is equally important because Copilot responds to the context supplied in code and natural-language comments. A specific task description, including constraints and intended behavior, gives it a clearer implementation target. Providing examples of desired output further strengthens that context by showing the expected result in a concrete form. Examples can clarify formatting, edge cases, return structures, and behavioral expectations that may not be fully conveyed by a brief description alone. Together, meaningful names, clear task definitions, and output examples make the prompt and code context more precise, enabling Copilot to generate more useful and aligned suggestions. GitHub recommends giving Copilot clear, specific context and using comments or natural language to describe the intended task. See GitHub's prompt-engineering guidance and GitHub Copilot best-practice guidance.

QUESTION NO: 13

Which GitHub Copilot organization-level policy can administrators configure to prevent Copilot from suggesting code that matches publicly available code?

A. GitHub Copilot Chat in the IDE

B. GitHub Copilot Chat in GitHub Mobile

C. GitHub Copilot suggestions matching public code filter

D. GitHub Copilot access to Bing

ANSWER: C

Explanation:

GitHub Copilot suggestions matching public code filter is the organization-level policy that controls whether Copilot provides code-completion suggestions that match code publicly available on GitHub. When an organization enables the blocking setting, Copilot compares eligible code-completion suggestions with public code and suppresses suggestions that meet GitHub's matching criteria. This gives organization owners a centralized governance control, so the policy is applied consistently to users covered by the organization's Copilot settings rather than depending on each developer's individual preference.

The feature was formerly called the *duplication detection filter*, so that wording may appear in older training material and documentation. GitHub now generally describes the setting as "suggestions matching public code." It applies specifically to Copilot code-completion behavior and is intended to help organizations manage the likelihood of receiving suggestions that substantially match public code. Administrators configure the relevant policy from their organization's Copilot policy settings. GitHub documents the available organization policies at Managing policies for Copilot in your organization and explains the public-code matching behavior at Managing suggestions from GitHub Copilot.

QUESTION NO: 14

Which types of textual content can GitHub Copilot Knowledge Bases use to answer questions? Each correct answer represents part of the solution. Choose three.

A. compiled binaries

B. code snippets

C. design patterns

D. screenshots

E. documentation

ANSWER: B C E

Explanation:

GitHub Copilot Knowledge Bases can ground Copilot's responses in relevant, searchable textual material, enabling it to provide answers that reflect the context supplied by an organization or project. **code snippets** are suitable because source code is text that Copilot can analyze to explain implementation details, usage, behavior, and relationships between components. **design patterns** are also appropriate subject matter: Copilot can identify and explain recurring architectural or implementation approaches when those patterns are represented in the available code and supporting written material. **documentation** is particularly valuable knowledge-base content because it commonly contains product guidance, setup instructions, APIs, conventions, and domain-specific information that users need to query. By retrieving relevant context from these textual sources, Copilot can produce answers that are more specific to the associated project rather than relying only on its general model knowledge. GitHub documents the use of knowledge bases to provide Copilot with context from curated content, and its Copilot documentation explains how repository and documentation context helps Copilot answer questions about a project. See Using GitHub Copilot knowledge bases and Responsible use of GitHub Copilot code completion.

QUESTION NO: 15

How can GitHub Copilot help maintain consistency throughout a project's test suite?

A. By identifying a pattern in the way you write tests and suggesting similar patterns for future tests.

B. By automatically fixing all tests in the code based on the context.

C. By providing documentation references based on industry best practices.

D. By writing the implementation code for the function based on context.

ANSWER: A

Explanation:

GitHub Copilot can help maintain consistency across a test suite by recognizing patterns in the code available in its context and proposing new test code that follows those established patterns. When a developer works in or near existing tests, Copilot can use surrounding code and open files to infer conventions such as the test framework in use, test naming style, setup and teardown approach, arrange-act-assert structure, mocking conventions, and assertion patterns. This makes it easier to create a new test that is stylistically and structurally aligned with the rest of the project, reducing repetitive effort while supporting readable, maintainable tests. Copilot does not independently apply changes: suggestions are presented to the developer, who remains responsible for reviewing, accepting, editing, or rejecting them and for validating that the test accurately verifies the intended behavior. GitHub documents that code completions are generated from context available in the editor, including code surrounding the cursor, which enables Copilot to reflect patterns already present in a repository. For more detail, see GitHub Copilot code completion concepts and Get code suggestions in your IDE with GitHub Copilot.

QUESTION NO: 16

Which two GitHub Copilot plans provide default data protections that exclude GitHub Copilot usage data, prompts, and suggestions from being used to train GitHub Copilot models?

A. GitHub Copilot Business

B. GitHub Copilot Codespace

C. GitHub Copilot Individual

D. GitHub Copilot Enterprise

ANSWER: A D

Explanation:

GitHub Copilot Business and GitHub Copilot Enterprise are GitHub's commercial Copilot plans and include a default commitment that GitHub will not use prompts, suggestions, or user-engagement data to train its models. This protection is

important for organizations whose developers may work with proprietary code, internal documentation, or other business-sensitive context while using Copilot. It applies as part of the commercial service's data-handling model, rather than depending on each individual user to change a training preference.

These plans are designed for organization-managed use and pair the data protection with centralized controls. Administrators can assign licenses and manage access for members of an organization or enterprise, supporting governance at scale. The Enterprise plan builds on these commercial protections while adding enterprise-oriented Copilot capabilities, and the Business plan provides the corresponding commercial privacy commitment for organization members. GitHub documents that it does not use Business and Enterprise prompts, suggestions, and user-engagement data for model training in its GitHub Copilot privacy statement. Plan availability and commercial-plan details are also described in the GitHub Copilot documentation.

QUESTION NO: 17

What risks can arise when developers rely too heavily on code generated by GitHub Copilot? Choose two answers.

A. GitHub Copilot may introduce security vulnerabilities by suggesting code with known exploits.

B. GitHub Copilot may decrease developer velocity by requiring too much time in prompt engineering.

C. GitHub Copilot's suggestions may not always reflect best practices or the latest coding standards.

D. GitHub Copilot may increase development lead time by providing irrelevant suggestions.

ANSWER: A C

Explanation:

GitHub Copilot may introduce security vulnerabilities by suggesting code with known exploits. Copilot generates suggestions from patterns learned from code and from the context available in the current workspace, but its output is not guaranteed to be secure or suitable for a specific application. Generated code can omit important input validation, authentication or authorization controls, error handling, and secure secret-management practices. Developers must therefore review, test, and scan Copilot-generated code before accepting it, just as they would code from any other source.

GitHub Copilot's suggestions may not always reflect best practices or the latest coding standards. A suggestion is not an authoritative decision about an organization's approved architecture, libraries, security requirements, or coding conventions. Recommendations can be incomplete, may not match the project's dependencies and constraints, and require validation against current team standards. Normal pull-request review, automated testing, linting, and security analysis remain essential when using generated code. GitHub explicitly states that users are responsible for reviewing and testing Copilot output before using it. See GitHub Copilot responsible use and responsible use of GitHub Copilot code completion.

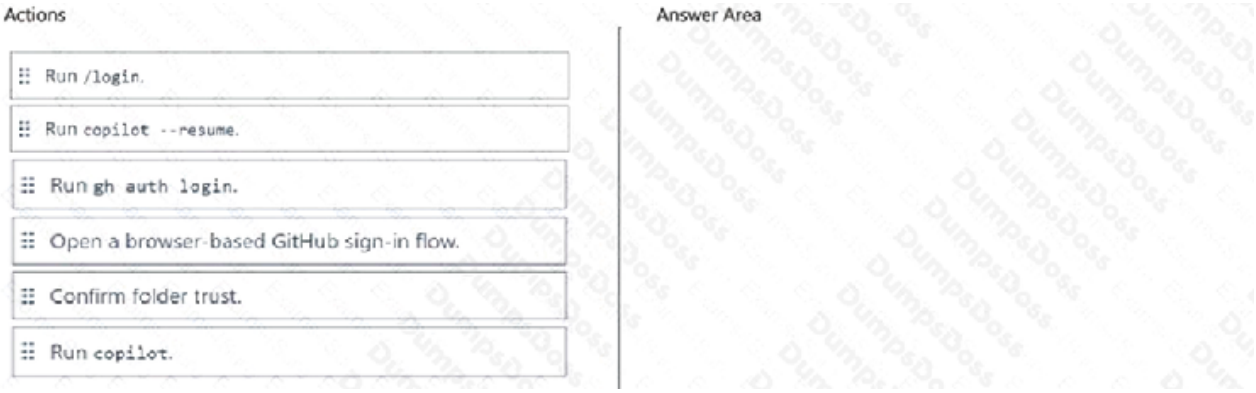
QUESTION NO: 18 - (DRAG DROP)

You have a GitHub Copilot subscription.

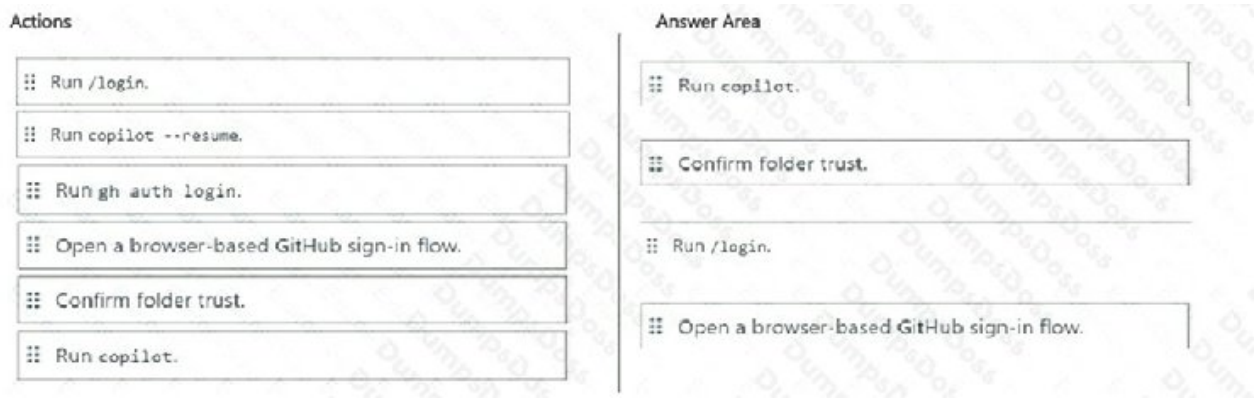
You need to start an interactive GitHub Copilot CLI session from the root folder of a project.

If you are NOT authenticated, GitHub Copilot must prompt you to sign in.

Which four actions should you perform in sequence? To answer, move the appropriate actions from the list of actions to the answer area and arrange them in the correct order.



ANSWER:



Explanation:

The correct sequence begins by running `copilot` while the terminal is located at the project's root folder. This starts the interactive GitHub Copilot CLI experience and makes the current directory the relevant workspace for the session. Starting from the root is important because it gives Copilot the appropriate project-level context when working interactively with files, commands, and repository content.

Next, confirm folder trust when Copilot requests it. Trust confirmation is the explicit local security approval that permits Copilot CLI to use the files in that directory as workspace context. Once the project is trusted, use `/login` within the interactive session. This slash command begins the Copilot CLI authentication process for a user who does not yet have an authenticated GitHub session.

The authentication flow then opens a browser-based GitHub sign-in page. In the browser, the user signs in to GitHub and authorizes the connection as required. After that authorization is completed, the CLI session is authenticated and can continue operating in the already trusted project folder. This sequence cleanly follows the interactive CLI workflow: enter Copilot in the intended workspace, approve that workspace, initiate login from within Copilot, and complete the browser authorization. GitHub documents starting and using the interactive CLI session in the GitHub Copilot CLI documentation. GitHub also describes browser-based authentication and authorization for GitHub-connected tools in its authentication documentation.

QUESTION NO: 19

How can GitHub Copilot and GitHub Copilot Chat help developers modernize an existing application?

A. GitHub Copilot can suggest modern programming patterns based on your code.

B. GitHub Copilot can create and deploy full-stack applications based on a single query.

C. GitHub Copilot can automatically refactor applications to align with upcoming standards.

D. GitHub Copilot can directly convert legacy applications into cloud-native architectures.

ANSWER: A

Explanation:

GitHub Copilot can suggest modern programming patterns based on your code. GitHub Copilot and Copilot Chat help with application modernization by understanding the code currently open in the developer's workspace and generating context-aware suggestions. A developer can ask Copilot Chat to explain older code, propose a more current implementation approach, generate replacement code, or help update code toward patterns and APIs appropriate for the project. For example, Copilot can assist with refactoring a method, improving readability, adopting a supported library API, adding tests, or producing code that follows an established framework pattern. These suggestions accelerate the developer's modernization work while keeping technical decisions, code review, validation, and deployment under the developer's control. GitHub describes Copilot as an AI coding assistant that provides code-completion suggestions and conversational assistance based on the active coding context. Its usefulness in modernization comes from helping developers identify and implement incremental improvements in an existing codebase, rather than performing an unsupervised end-to-end transformation. See What is GitHub Copilot? and Getting code suggestions in your IDE.