GitHub Copilot入門

Section 02

このセクションで学ぶこと

学習目標

- GitHub Copilotの基本概念と役割を理解する
- GitHub Copilotの起動方法と操作方法を習得する
- Copilotとの効果的な対話方法を学ぶ
- 設定・構成・トラブルシューティングを把握する

GitHub Copilot認定試験の基礎となる重要なセクションです

Lecture 1: GitHub Copilotとは

GitHub Copilotの概要

GitHub Copilotとは？

- **定義**: OpenAIと共同開発されたAIペアプログラマサービス
- **技術基盤**: OpenAI Codexシステム（GPT-3より優れたコード生成能力）
- **機能**: オートコンプリート形式でリアルタイムにコード候補を提案

対応プログラミング言語

Python、JavaScript、TypeScript、Ruby、Go、C#、C++ など多数

試験ポイント: 技術基盤は「OpenAI Codex」です

対応IDE

GitHub Copilotが使えるIDE

IDE	説明
Visual Studio Code	最も人気のある軽量エディター
Visual Studio	Microsoft製の統合開発環境
Vim / Neovim	ターミナルベースのエディター
JetBrains スイート	IntelliJ、PyCharm等

試験ポイント: 主要なIDEはすべてサポートされています

GitHub Copilotの効果（調査データ）

開発者への影響

指標	数値
AIによって記述された新しいコードの割合	**46%**
開発者の生産性向上	**55%**
満足のいく作業に集中できると感じた開発者	**74%**

試験ポイント: この3つの数値は頻出です。必ず覚えましょう！

- 46% → AIが書いたコード
- 55% → 生産性向上
- 74% → 満足度

GitHub Copilotの主要機能

4つの主要機能

機能	説明
コード補完	インラインでリアルタイムにコード提案
チャット用Copilot	ChatGPTライクなインターフェイス
Pull Request用Copilot	PR説明の自動生成（GPT-4モデル使用）
CLI用Copilot	ターミナルでのコマンド構成支援

試験ポイント:「自動デプロイ機能」は含まれません

AIペアプログラマとは

概念の理解

- 開発者と協働してコーディングを支援するAI
- コンテキストを理解して適切な提案を生成
- 単なるオートコンプリートではなく、**複数行のコード提案が可能**

ポイント

GitHub Copilotは「AIペアプログラマ」として、開発者の横で一緒にコーディングする存在です

Lecture 2: GitHub Copilotの起動方法

トリガー方法

GitHub Copilotを起動する3つの方法

方法	説明
自動トリガー	コードを入力すると自動的に提案が表示される（淡色表示）
手動トリガー	キーボードショートカットで明示的に起動
コメント駆動	自然言語のコメントを書いてコードを生成

試験ポイント: 「ファイル保存で自動最適化」はトリガー方法ではありません

コマンドパレットからの起動

起動方法

- **Windows/Linux**: `Ctrl + Shift + P`
- **Mac**: `Cmd + Shift + P`

手順

1. コマンドパレットを開く
2. 「Copilot」と入力
3. 使用可能なコマンドを表示

提案の操作（キーボードショートカット）

覚えておくべきショートカット

操作	Windows/Linux	Mac
提案を受け入れる	`Tab` または `→`	`Tab` または `→`
提案を拒否する	`Esc`	`Esc`
次の提案を表示	`Alt + ]`	`Option + ]`
前の提案を表示	`Alt + [`	`Option + [`

試験ポイント: 提案を受け入れるのは「Tab」または「→」キーです

Lecture 3: Copilotとの対話方法

効果的な対話のポイント

より良い提案を得るために

ポイント	説明
明確なコンテキスト	変数名や関数名を意味のあるものにする
コメントの活用	やりたいことを自然言語で説明する
段階的なアプローチ	複雑な処理は小さなステップに分ける

試験ポイント: 「短い変数名」や「コメントなし」は推奨されません

コメント→コード変換

自然言語からコードを生成

コメントを書くだけでコードを生成できる：

```
# Function to reverse a string
def reverse_string(s):
    return s[::-1]
```

プロンプト（入力）は、GitHub Copilotに何を生成するかを伝える指示です

Copilotチャット

ChatGPTライクな対話型インターフェイス

- 自然言語で対話型コミュニケーション
- 質問やコードスニペットをリクエスト可能
- 新しい概念の探索や構文サポートに最適

活用例

- 「Python でバイナリ検索を実装するには？」
- 「このコードの意味を説明して」
- 「このエラーの原因は？」

インラインチャット

エディター内でコンテキスト固有の会話

項目	内容
起動方法（Windows/Linux）	`Ctrl + I`
起動方法（Mac）	`Cmd + I`

特徴

- コードエディター内で直接対話
- コンテキストを切り替えずに質問・変更を依頼

試験ポイント: インラインチャットは「Ctrl + I」で起動します

スラッシュコマンド

よく使うスラッシュコマンド

コマンド	説明
`/explain`	選択したコードの説明
`/suggest`	コードの提案
`/tests`	単体テストの生成
`/comment`	コメントからコードを生成

試験ポイント: 単体テスト生成は「/tests」コマンドです

説明機能とテスト生成

コードの説明

1. コードを選択
2. 右クリック → 「Copilot: Explain This」
3. 選択したコードの説明を取得

自動化されたテスト生成

```
def add(a, b):
    return a + b

# Copilotが生成するテスト例
def test_add():
    assert add(2, 3) == 5
    assert add(-1, 1) == 0
```

Copilotの学習特性

理解しておくべきポイント

- コンテキストから学習する
- コードの適切な構造化とコメント記載が精度を向上させる
- 相互作用を重ねるほどコーディングスタイルに適応する

試験ポイント: 出力の品質は「プロンプトをどの程度適切に作成したか」で決まります

Lecture 4: 設定と構成

GitHub Copilotへのサインアップ

サインアップ手順

1. GitHubのプロフィール写真を選択 → **設定**
2. 左メニューの「コード、計画、自動化」から **Copilot** を選択
3. 無料試用版またはサブスクリプションを設定

プラン

- **GitHub Copilot Free**: 月2,000コード補完、50チャット要求
- **GitHub Copilot Pro**: 無制限のコード補完とチャット
- **GitHub Copilot Business**: 組織向け管理機能付き
- **GitHub Copilot Enterprise**: 大規模組織向け高度機能

VS Codeでの拡張機能インストール

インストール手順

1. Visual Studio Marketplaceで「GitHub Copilot」を検索
2. **インストール** を選択
3. VS Codeが開いたら拡張機能タブで **インストール** を選択
4. 初回はGitHubにサインインが必要

試験ポイント: VS Code以外にも Visual Studio、JetBrains IDE、Neovim がサポートされています

有効/無効の切り替え

操作方法

1. VS Code下部の状態アイコンをクリック
2. **有効にする** または **無効にする** を選択

無効化のオプション

- グローバルに無効化
- 特定の言語に対して無効化

インライン候補の設定

ファイル → **ユーザー設定** → **設定** → **拡張機能** → **GitHub Copilot**

組織での管理

管理者向け機能

- Organization/Enterpriseでのポリシー設定
- ユーザーへのライセンス割り当て
- コンテンツ除外の設定

GitHub Copilot Business/Enterprise

- 一元管理とポリシー制御
- セキュリティの脆弱性のフィルター処理
- コード参照とパブリックコードのフィルター処理

Lecture 5: トラブルシューティング

よくある問題と解決方法

一般的なトラブル

問題	解決方法
提案が表示されない	拡張機能の有効化を確認、ネットワーク接続を確認
認証エラー	GitHubアカウントの再認証
遅延が発生	ネットワーク状況の確認、プロキシ設定の見直し

一般的な原因

- ネットワーク制限
- ファイアウォール設定
- プロキシ接続の問題

試験ポイント: 「言語がサポートされていない」は一般的な原因ではありません

ログファイルと診断情報

ログファイルの確認

コマンドパレット（`Ctrl+Shift+P` / `Cmd+Shift+P`）で：

- **「開発者: ログ ファイルを開く」**
- **「開発者: 拡張機能のログ フォルダーを開く」**

Electronログの表示

ヘルプ → 開発者ツールの切り替え

診断情報の収集

1. コマンドパレットを開く
2. 「Diagnostics」と入力
3. **「GitHub Copilot: 診断情報の収集」** を選択

まとめ

このセクションで学んだこと

#	トピック	内容
1	GitHub Copilotの基本	AIペアプログラマとしての役割と機能
2	起動方法	自動・手動・コメント駆動のトリガー方法
3	対話方法	効果的なコンテキスト提供とインラインチャット
4	設定	個人および組織での構成方法
5	トラブルシューティング	一般的な問題の解決方法

試験対策：重要ポイント

数値を覚えよう

- **46%**: AIによって記述された新しいコードの割合
- **55%**: 開発者の生産性向上
- **74%**: 満足のいく作業に集中できると感じた開発者

キーボードショートカット

- 提案を受け入れる: `Tab` または `→`
- インラインチャット: `Ctrl + I` / `Cmd + I`
- 提案を拒否: `Esc`

これらは試験で頻出です！

試験対策：よくある誤答

以下は正しくない選択肢です

- 「自動デプロイ機能」は GitHub Copilot の機能ではない
- 「ファイル保存で自動最適化」はトリガー方法ではない
- 「短い変数名」は推奨されていない
- 「Rust」は「強力なサポート言語」として言及されていない

誤答の選択肢を見分けられるようにしましょう

Section 02 完了

次のセクション: 責任あるAI

お疲れさまでした！