最近のプログラミング環境（Github，Github Copilot等）の利活用（ダイジェスト版）

本資料は，VS Code を用いた Git によるバージョン管理，GitHub を使用したチーム開発，GitHub Copilot による AI 支援開発の3つの実習内容をまとめたダイジェスト版です．

これらの実習では，Gitによるコードのバージョン管理の基礎を体験した後，グループでリーダーがリモートリポジトリを管理し，メンバーがそれぞれブランチを作成して機能拡張を行い，プルリクエストを通じて変更を提案，リーダーが確認した上で採用するという，実際のソフトウェア開発の流れを体験します．さらに，GitHub Copilot という AI 支援開発ツールを活用したコード補完も体験します．

VS Code での Git/GitHub によるバージョン管理

はじめに

Visual Studio Code（VS Code）は，Microsoft が開発した軽量で高機能な統合開発環境です．Git のサポートが標準で組み込まれており，直感的な GUI 操作でバージョン管理を行うことができます．本実習では，ローカルでの Git 操作と，GitHub を使ったリモートリポジトリとの連携の両方を体験します．

Git と GitHub

Git は，プログラムのソースコードなどの変更履歴を記録・追跡するための分散型バージョン管理システムです．GitHub は，Git の機能を基盤としたクラウドベースのホスティングサービスで，リモートリポジトリを通じてチームでの協働開発を可能にします．Git と GitHub を使用することで次のことが可能になります：

- **変更履歴の記録**：いつ誰がどのような変更をしたか，すべてが記録される
- **過去のバージョンへの復帰**：バグが導入された場合，以前のバージョンに戻すことができる
- **複数人での並行開発**：ブランチを使うことで，複数のメンバーが同時に異なる機能を開発できる
- **コンフリクト解決**：異なるメンバーが同じ部分を編集した場合，競合を解決する仕組みがある

ファイルを扱うための4つの領域

領域名	説明	利用
ワークスペース	現在編集しているファイルのある場所．変更はまだ Git に記録されていない．	Git
ステージング領域	「次のコミットに含める」と指定したファイルが置かれる場所．	Git
ローカルリポジトリ	コンピュータ内に保存された変更履歴の記録．プロジェクト内の `.git` フォルダ．	Git
リモートリポジトリ	GitHub 等のサーバー上に保存されたプロジェクトの変更履歴．チームで共有．	GitHub

主要な操作

操作	流れ	説明	利用
Add（ステージング）	ワークスペース → ステージング領域	変更を次のコミットに含める意思表示	Git
Commit（コミット）	ステージング領域 → ローカルリポジトリ	スナップショットとして保存	Git
Push（プッシュ）	ローカル → リモートリポジトリ	変更をサーバーにアップロード	GitHub
Pull（プル）	リモート → ローカルリポジトリ	最新の内容をダウンロード	GitHub

ブランチの概念

ブランチは，メインの開発線から分岐させた独立した作業領域です．自身で追加機能ごとにブランチを分けたり，複数の開発者が異なる機能を同時に開発できるようにするための仕組みです．例えば次のような名前でメインのブランチと開発用のブランチを扱うことを考えます．

- **main（master）ブランチ**：本番環境にリリース可能な安定版
- **feature/新機能名ブランチ**：新機能開発用の一時的なブランチ

VS Code での Git 操作手順

1. リポジトリの初期化

Source Control パネル（ `Ctrl+Shift+G` ）を開き，「リポジトリの初期化」ボタンをクリックします．

2. ステージング

変更ファイルの右側に表示される「＋」ボタンをクリックするか，右クリックで「ステージ」を選択します．ステージされたファイルは「ステージされている変更」セクションに移動します．

3. コミット

テキスト入力欄にコミットメッセージ（例：「初期コミット」）を記述し，「✓」ボタンまたは `Ctrl+Enter` でコミットします．

4. ブランチの作成と切り替え

ステータスバー左下のブランチ名をクリックし，「+新しいブランチの作成」を選択します．ブランチ名は `feature/reset-button` のように機能を示す名前にします．

5. マージ

マージ先のブランチに切り替えた状態で，「表示 → コマンドパレット」から「Git: マージ」を選択し，マージ元のブランチを指定します．競合がある場合は解決が必要です．

GitHub 連携とチーム開発

上でGit の知識を応用して，GitHub 上でのリモートリポジトリを利用したチーム開発の流れを体験します．題材として，tkinter で作成したカウンタアプリを使用し，4名ずつのグループでリーダーがリポジトリを管理し，メンバーがそれぞれ機能拡張（ボタンの追加，表示の変更など）を行います．

GitHub の基礎概念

GitHub とは

GitHub は，Git の機能を基盤として構築されたクラウドベースのホスティングサービスです．世界中の開発者が，プロジェクトのコードを保存し，共有し，協働開発するためのプラットフォームとして使用されています．Microsoft 傘下の企業で運営されており，現在，世界で最も広く使用されるコード管理プラットフォームです．

リモートリポジトリとは

「ローカルリポジトリ」は皆さんのコンピュータ内に保存されたプロジェクトの履歴です．これに対して，「リモートリポジトリ」は GitHub 上のサーバーに保存されたプロジェクトの履歴です．リモートリポジトリを使用することで，複数のメンバーがインターネット経由でアクセスでき，同じプロジェクトに対して協働作業ができます．

プルリクエスト（Pull Request）とは

「自分のブランチで行った変更を，メインブランチにマージしてほしい」と提案するための仕組みです．リーダーやチームメンバーが内容を確認した上で承認するプロセスがあり，コード品質の維持やバグの早期発見に役立ちます．

フォークとクローン

- **クローン**：リモートリポジトリの内容をローカルマシンにコピー（メンバーとして招待されたリポジトリ向け）
- **フォーク**：他人の公開リポジトリを自分の GitHub アカウントにコピー（権限のないプロジェクト向け）

VS Code での GitHub 設定

GitHub 認証

VS Code 左のバーの人のアイコンから「バックアップと同期の設定」→「サインイン」→「GitHub でサインイン」を選択します.

ローカル Git 設定の確認

```
git config --global user.name
git config --global user.email
```

設定されていない場合は以下で設定：

```
git config --global user.name "Your Name"
git config --global user.email "your.email@example.com"
```

チーム開発の例

リーダーの役割

1. ローカルリポジトリを「GitHub に発行」機能でリモート化し，メンバーを招待
2. main ブランチをリモートにプッシュして公開
3. メンバーからのプルリクエストを確認・検討
4. 問題がなければマージ，必要に応じてフィードバック

メンバーの役割

1. リモートリポジトリをクローンしてローカルに取得
2. main ブランチから自分のフィーチャーブランチを作成（例： `feature/member1_extension` ）
3. 割り当てられた機能拡張を実装
4. コミットメッセージを記入してコミット
5. フィーチャーブランチをリモートにプッシュ
6. GitHub 上でプルリクエストを作成

体験してみよう

リーダー：GitHub への発行

Source Control パネルの「...」メニューから「GitHub に発行」を選択．リポジトリ名（例： `g03_counter_app` ），説明，公開範囲（Private）を設定し，「Create private repository」をクリックします.

リーダー：メンバーの招待

GitHub のリポジトリページで「Settings」→「Collaborators」→「Add people」からメンバーを招待します.

メンバー：クローンとブランチ作成

コマンドパレット（ `Ctrl+Shift+P` ）から「Git: クローン」→「Github から複製」を選択し，リポジトリを選んでクローンします．その後，フィーチャーブランチを作成して「ブランチの発行」を行います.

メンバー：プルリクエスト作成

変更をプッシュ後，GitHub のリポジトリページで「Compare & pull request」ボタンをクリックし，タイトルと説明を記入して「Create Pull Request」を実行します.

リーダー：確認とマージ

「Pull Requests」タブでプルリクエストを確認し，問題がなければ「Merge pull request」をクリックしてマージします．

（参考）カウンタアプリの拡張例

各メンバーで異なる拡張を選択して実装します：

- ボタンテキストの変更（「へぇー」→「カウント」「+1」など）
- ウィンドウタイトルの変更
- ボタンの色変更（ `fg="blue"` など）
- +10 ボタンの追加
- カウント初期値の変更
- ラベルテキスト形式の変更

GitHub Copilot での AI 支援開発

はじめに

これまでの実習で学んだ VS Code での Git 基本操作と GitHub を使用したチーム開発の知識に加えて，GitHub Copilot という AI 支援開発ツールを体験します．題材として引き続き tkinter のカウンタアプリを使用し，Copilot の支援を受けながら新機能（カウント履歴の保存・表示，ランダムボタンの追加など）を実装します．

（重要）Copilot が生成したコードは必ず自分で目を通し，正確性やセキュリティを確認することが不可欠です．AI は便利ですが，最終的な責任は開発者自身にあります．

GitHub Copilot の概要

Copilot とは

GitHub Copilot は，Microsoft が提供する AI コード補完サービスです．大規模なコードデータセットで学習した GPT-4 ベースのモデルが，開発者が書いているコードの文脈を理解し，適切な補完提案をリアルタイムで提供します．Python，JavaScript，Java，C++ など複数のプログラミング言語に対応しています．

2つの主要機能

機能	説明
Copilot（自動補完）	コード入力中に自動的に補完案を表示．Tab キーで受け入れ，Esc キーで却下．
Copilot Chat（チャット機能）	自然言語で質問を入力し，AI が詳細な説明やコード生成を行う．

セットアップ

拡張機能のインストール

1. VS Code の拡張機能（四角いマーク）をクリック
2. 「GitHub Copilot」を検索してインストール
3. 「GitHub Copilot Chat」も同様にインストール
4. VS Code を再起動

GitHub アカウント認証

初めて使用する際に GitHub アカウントの認証が求められます．右下の Copilot アイコンをクリックすると使用状況が確認できます．

Copilot の活用方法

コメントからの補完（カウンタアプリでの例）

カウンタアプリの `counter_app.py` を開き，コメントを入力すると Copilot がグレーで提案を表示します．例えば `__init__` メソッドの最後に：

```
# ランダムに値を変えるボタン
```

と入力すると，ボタン生成コードとランダム処理のメソッドが提案されます．Tab キーで受け入れ，Esc キーで却下します．

複数提案の比較

`Ctrl+Enter`（または `Alt+]`）で複数の提案案から選択できます．

Copilot Chat の利用

アクティビティバーの「Copilot Chat」アイコン（または `Ctrl+Shift+I`）でチャットパネルを開き，自然言語で質問できます．カウンタアプリの開発で役立つ質問例：

質問例	説明
Tkinter で複数のボタンを水平方向に配置するには？	カウンタアプリのボタンレイアウト改善
カウント値を JSON ファイルに保存する方法を教えてください	カウント履歴機能の実装
このコードにエラーハンドリングを追加してください	選択したコードの改善案と説明を提示

Chat のモード

チャットには **Ask**，**Edit**，**Agent** の3つのモードがあります．Agent モードでは，やり取りの中でエージェントがコードの改変を提案・実行します．

活用のコツと注意点

効果的なコメントの書き方

曖昧なコメント（「処理を追加」）よりも，具体的なコメント（「カウント値がゼロ以上100以下の整数であることを確認する検証関数を追加」）の方が，より適切なコードが生成されます．

コードレビューの重要性

Copilot が生成したコードは，必ず自分で目を通して確認（レビュー）してください．エラーハンドリングの不足，パフォーマンスの問題，セキュリティ上の懸念がないかを確認します．

Copilot の限界

- 生成されたコードにバグが含まれる可能性
- ライセンスが不明瞭なコードが提案されるリスク
- 古い実装パターンが提案されることもある

なお，大学生は GitHub Education から申請することで，「GitHub Copilot Pro」を無料で利用できます．学校発行のメールアドレスや学生証等が必要です．

GitHub CopilotのようなAIツールは今後も様々なものが登場し，便利になってくると思われます．一方で，上記のような問題があったり，学生として学ぶことと利用することとのバランスに悩んだりすることがあると思います．その際，皆さんは，その利活用が「自分の可能性を拡げる」ことにつながるかを常に意識しながら活用してほしいと思っています．