AI プログラミング実践ガイド：実験と探求

第 1 章 環境構築

本章の目標は、自身のパソコンで AI プログラミングの実行と実験が可能な環境を構築することである。本章を完了すると、さまざまな AI プログラムを動作させたり、ソースコードの確認や修正ができる環境が整う。

1.1 Python の役割

Python は AI プログラミングにおいて広く利用されているプログラミング言語である。**機械学習**や**ディープラーニング**のライブラリが豊富であり、初学者にも扱いやすい。

1.2 AI プログラミングにおける開発環境の全体像

本章では以下のソフトウェアをインストールする。

- **Python 本体**：プログラミング言語
- **7-Zip**：圧縮・展開を行うツール
- **Git**：バージョン管理ツール。GitHub などのサイト上で公開されている多くの AI プロジェクトのファイルを取得する際に使用する
- **Visual Studio 2022 Build Tools とランタイム**：一部の Python ライブラリが C++で記述された拡張機能を必要とするため、C++コンパイラと関連ライブラリを提供する
- **NVIDIA CUDA**：NVIDIA GPU 上で並列計算を実行するためのプラットフォーム。**GPU（Graphics Processing Unit）**は並列計算能力を持つプロセッサであり、ディープラーニングの高速化に活用される
- **PyTorch**：研究分野で広く使用される深層学習フレームワーク

- **開発エディタ**：コードの作成と編集を行うツール（例：Windsurf、Cursor、Visual Studio Code）

1.3 実行前の確認事項

- **ディスク空き容量が 20GB 以上あることを確認する**
- すでに **Python 3.12** がインストールされている場合は、そのまま使用できる。異なるバージョンの場合は、1.4 節の手順で Python 3.12 をインストールする

1.4 ソフトウェアのインストール（Windows）

Windows では、**winget** という Windows の公式パッケージ管理ツールを用いて多くのソフトウェアをインストールできる。**管理者権限で実行し、システム全体にインストールすることを推奨する。**

管理者権限でコマンドプロンプトを起動（手順：Windows キーを押して cmd と入力し、右クリックで「管理者として実行」を選択）し、以下を実行する。


```
REM Python をシステム領域にインストール
winget install --scope machine --id Python.Python.3.12 -e --silent --accept-source-agreements --
accept-package-agreements

REM パス長制限の解除
reg add "HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Control¥FileSystem" /v
LongPathsEnabled /t REG_DWORD /d 1 /f
```

```
reg query "HKLM¥SYSTEM¥CurrentControlSet¥Control¥FileSystem" /v LongPathsEnabled

REM Python のパス設定
set "PYTHON_PATH=C:¥Program Files¥Python312"
set "PYTHON_SCRIPTS_PATH=C:¥Program Files¥Python312¥Scripts"
echo "%PATH%" | find /i "%PYTHON_PATH%" >nul
if errorlevel 1 setx PATH "%PYTHON_PATH%;%PATH%" /M >nul
echo "%PATH%" | find /i "%PYTHON_SCRIPTS_PATH%" >nul
if errorlevel 1 setx PATH "%PYTHON_SCRIPTS_PATH%;%PATH%" /M >nul


REM 7-Zip をシステム領域にインストール
winget install --scope machine --id 7zip.7zip -e --silent --accept-source-agreements --accept-package-agreements

REM 7-Zip のパス設定
set "SEVENZIP_PATH=C:¥Program Files¥7-Zip"
if exist "%SEVENZIP_PATH%" echo "%PATH%" | find /i "%SEVENZIP_PATH%" >nul
if exist "%SEVENZIP_PATH%" if errorlevel 1 setx PATH "%PATH%;%SEVENZIP_PATH%" /M >nul

REM Git をシステム領域にインストール
winget install --scope machine --id Git.Git -e --silent --accept-source-agreements --accept-package-agreements

REM Git のパス設定
set "GIT_PATH=C:¥Program Files¥Git¥cmd"
if exist "%GIT_PATH%" echo "%PATH%" | find /i "%GIT_PATH%" >nul
if exist "%GIT_PATH%" if errorlevel 1 setx PATH "%PATH%;%GIT_PATH%" /M >nul

REM Visual Studio 2022 Build Tools とランタイムのインストール
winget install --scope machine --wait --accept-source-agreements --accept-package-agreements Microsoft.VisualStudio.2022.BuildTools Microsoft.VCRedist.2015+.x64

REM インストーラーとインストールパスの設定
set VS_INSTALLER="C:¥Program Files (x86)¥Microsoft Visual Studio¥Installer¥vs_installer.exe"
set VS_PATH="C:¥Program Files¥Microsoft Visual Studio¥2022¥BuildTools"

REM C++開発ワークロードのインストール（以下のコマンドは改行されているが、実行時は 1 行で入力する必要がある）
```

```
%VS_INSTALLER% modify --installPath %VS_PATH% --add Microsoft.VisualStudio.Workload.VCTools --
add Microsoft.VisualStudio.Component.VC.Tools.x86.x64 --add
Microsoft.VisualStudio.Component.Windows11SDK.22621 --includeRecommended --quiet --norestart

REM CUDA をシステム領域にインストール
winget install --scope machine --accept-package-agreements --accept-source-agreements -e
Nvidia.CUDA Microsoft.VCRedist.2015+.x64
REM CUDA のパス設定
set "CUDA_PATH=C:¥Program Files¥NVIDIA GPU Computing Toolkit¥CUDA¥v12.6"
if exist "%CUDA_PATH%" setx CUDA_PATH "%CUDA_PATH%" /M >nul
if exist "%CUDA_PATH%" setx CUDNN_PATH "%CUDA_PATH%" /M >nul

REM PyTorch をインストール
pip install -U torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126

REM Windsurf をシステム領域にインストール
winget install --scope machine --id Codeium.Windsurf -e --silent --accept-source-agreements --
accept-package-agreements

REM Microsoft VS Code をシステム領域にインストール
winget install --scope machine --id Microsoft.VisualStudioCode -e --silent --accept-source-agreements
--accept-package-agreements
if exist "C:¥Program Files¥Microsoft VS Code¥bin" cd "C:¥Program Files¥Microsoft VS Code¥bin"
if exist "C:¥Program Files¥Microsoft VS Code¥bin" code --install-extension ms-python.python
if exist "C:¥Program Files¥Microsoft VS Code¥bin" code --install-extension ms-python.vscode-pylance
if exist "C:¥Program Files¥Microsoft VS Code¥bin" code --install-extension MS-CEINTL.vscode-
language-pack-ja
if exist "C:¥Program Files¥Microsoft VS Code¥bin" code --install-extension dongli.python-preview
if exist "C:¥Program Files¥Windsurf¥bin" windsurf --install-extension MS-CEINTL.vscode-language-
pack-ja
```

実行時のヒント

- コマンドの途中で止まった場合は、まず **Enter キーを 1 回押し、エラー内容を確認する**
- **Python のインストールのときに「Modify Repair Uninstall」画面が出た場合は、Python がインストール済みなので「Cancel」をクリックする**

第 2 章 Python プログラム実行による探求

第 1 章で構築した環境を使用して、実際に AI プログラムを実行し、**パラメータ変更による効果を観察する**。本章では、**pip** を使用した Python ライブラリのインストール方法と、プログラムの変更および再実行による探求手法を習得する。

探求

本章における**「探求」**は、**プログラムのパラメータを変更し、その結果を観察・考察するサイクル**を指す。

2.1 Windsurf でのプログラム実行

Windsurf の初期設定

1.**コマンドプロンプト**を開き、python --version を実行して Python 実行環境の確認を行う

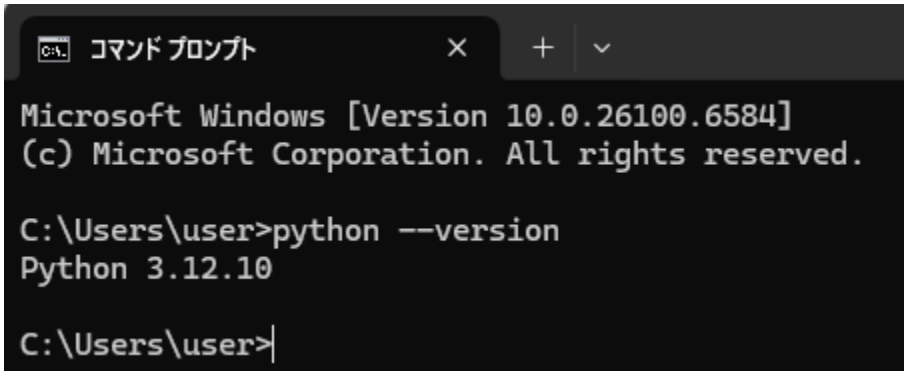


図. Python のバージョン確認

2. スタートメニューから **Windsurf** を起動する

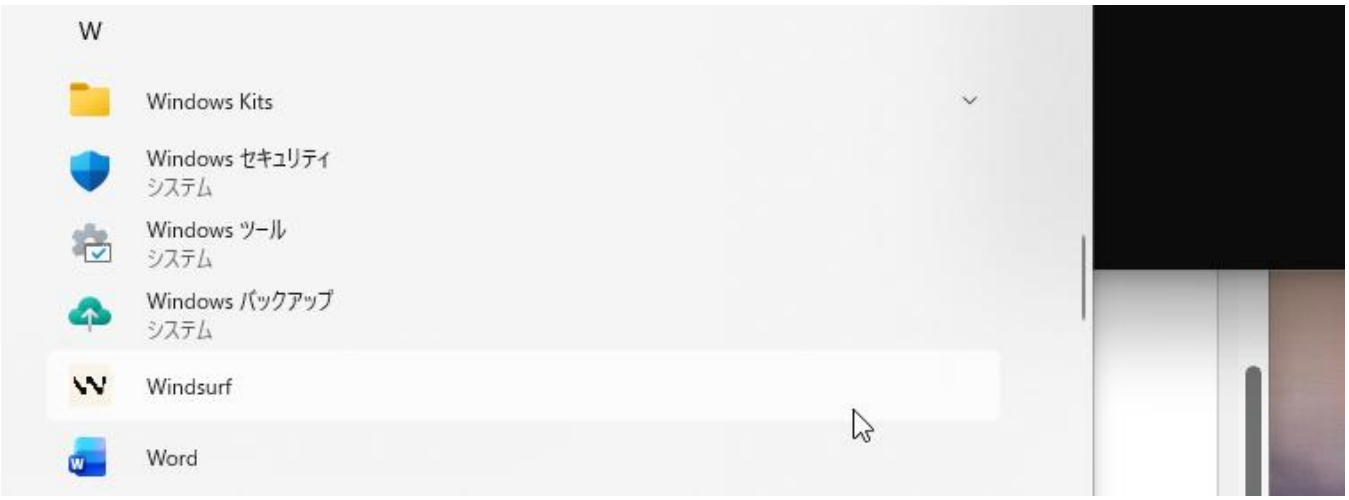


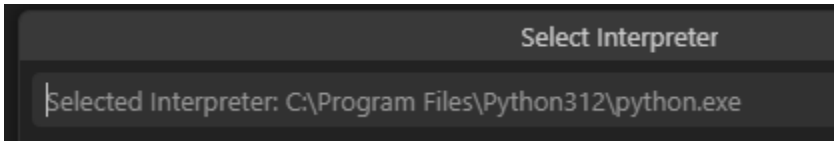
図. スタートメニュー

3. 初回起動時に**アカウント作成またはログイン**が求められた場合には、指示に従う

　アカウント作成やログインをしなくても使い続けることはできるが、機能や利用できる量に制限がかかる。

4. Python 実行環境の設定を確認する。**Ctrl+Shift+P** を押し、「Python: Select Interpreter」を選択する。

下図のように「C:¥Program Files¥Python312¥python.exe」と表示されれば、システムにインストールされた Python 3.12 が選択されている。選択されていなければ変更する



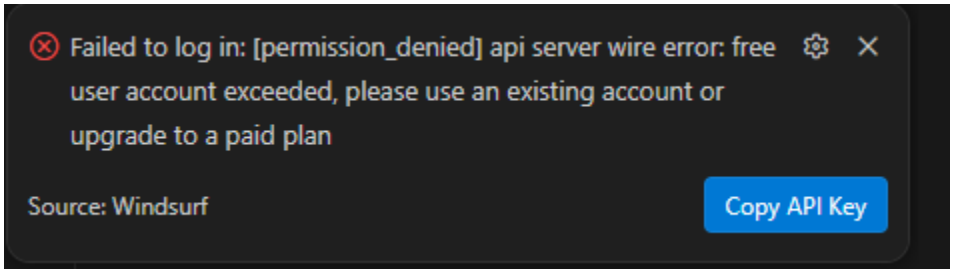
Windsurf の AI 支援機能

Windsurf は、AI を活用したプログラミング支援機能を提供する開発エディタである。主な機能は以下の通りである。これらの機能により、コーディング作業を効率化し、プログラミング学習を支援できる。

- **Cascade**：自然言語での指示に基づき、AI がコードの生成や編集を行う。Ctrl+L で起動する
- **インライン編集**：コード編集中に、**Ctrl+スペース**で AI による補完や修正の提案を受けられる
- **チャット機能**：プログラミングに関する質問や、コードの説明を対話形式で確認できる

実行中の警告表示

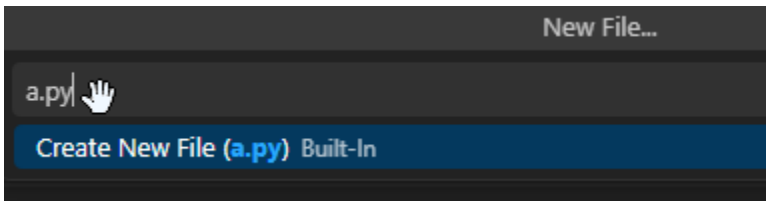
ログインせずに利用する場合には次の表示が出る場合がある。その際は「Copy API Key」をクリック。



新規 Python ファイルの作成と実行

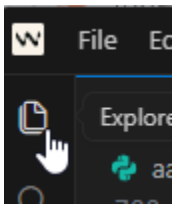
1. メニューから **File → New File** を選択する

2. ファイル名を入力する（例：a.py）。Enter キー



3. 保存したいディレクトリを選び、**Create File** をクリックする

4. サイドバーのファイルエクスプローラーから既存のファイルを開くこともできる



5. 作成したファイルに print("Hello")と入力し、エディタ上部の**実行ボタン**をクリックして実行する。
実行結果は画面下部のターミナルまたは出力ウィンドウに表示される

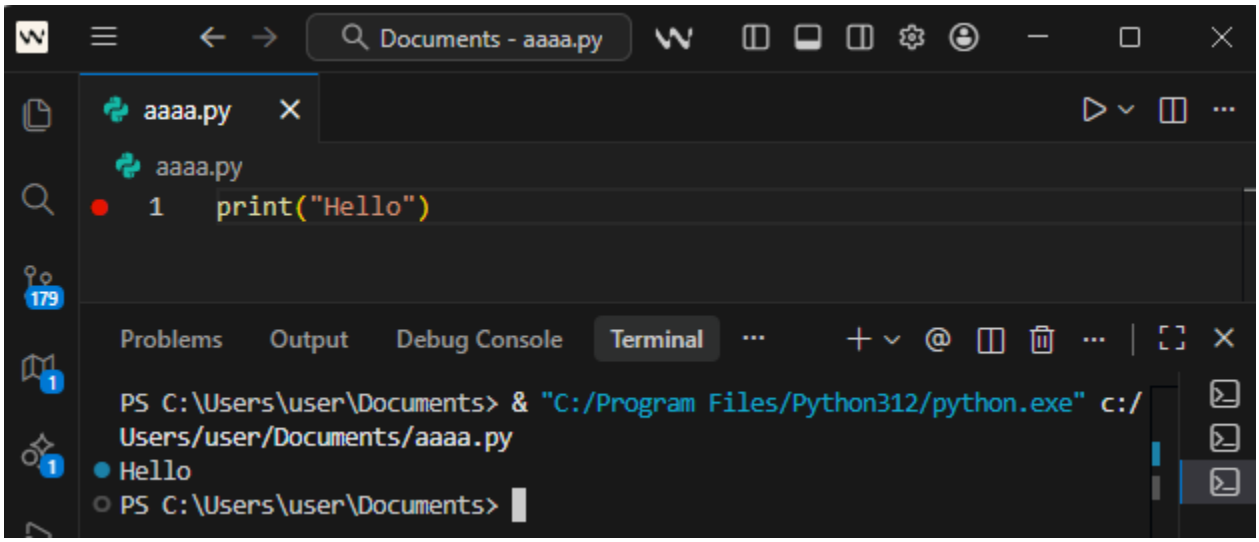


図 Windsurf 実行画面

主な操作

- Cascade チャット起動：**Ctrl+L**
- インライン編集：**Ctrl+スペース**
- デバッガー：**F5**
- プログラミング言語の選択：**Ctrl+K M**

Python のバージョン、Git のバージョン、PyTorch のバージョン、GPU の動作の確認

次のプログラムを Windsurf で実行し、Python のバージョン、Git のバージョン、PyTorch のバージョン、GPU の動作を確認する。主要なソフトウェア（Python、Git、PyTorch、GPU）のインストールに問題がないかを確認できる。

```
import sys
import subprocess
import torch

print(sys.version)
print(subprocess.check_output(['git', '--version'], text=True).strip())
print(torch.__version__)
print(torch.cuda.is_available())
```

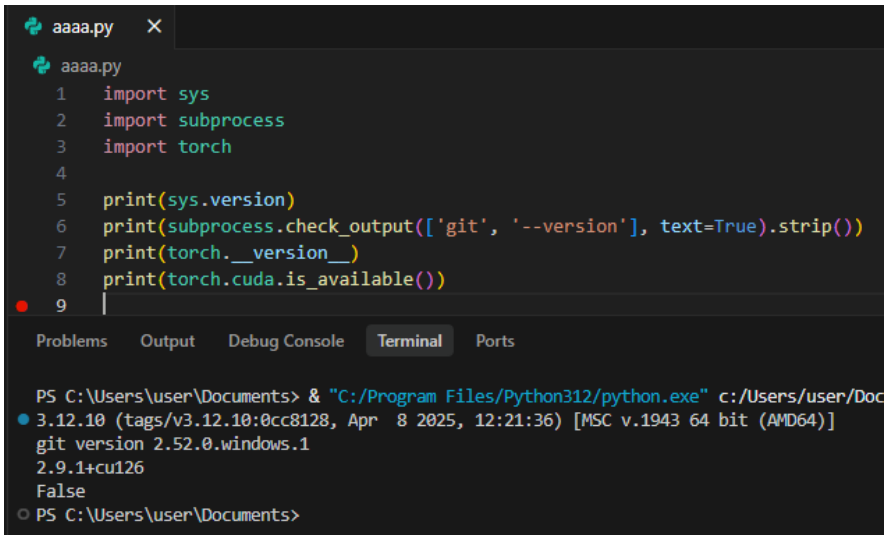


図. 実行結果例（GPU を搭載していないパソコンの場合、4 つ目の結果は False）

2.2 必要なライブラリのインストール

Python プログラムはそれぞれ異なるライブラリを必要とする（例：後述のプログラムでは **matplotlib** ライブラリが必要である）。必要なライブラリがインストールされていない場合は、以下の手順でインストールする。

手順

1. **管理者権限でコマンドプロンプトを起動する**。Windows キーを押して cmd と入力し、右クリックで「管理者として実行」を選択する

2. **pip** コマンドでライブラリをインストールする。必要なライブラリはプログラムによって異なる。以下は代表的な例である。

```
pip install numpy pandas matplotlib pillow requests
```

3. インストール完了後、正常に終了したことを確認する

主要な Python ライブラリ

Python には、様々な分野で活用できるライブラリがある。機械学習やデータサイエンス分野では、以下のライブラリが使用される。

- **PyTorch**：Meta が開発したオープンソースの機械学習ライブラリである。深層学習モデルの構築と訓練に使用され、柔軟性と高速な GPU 計算をサポートする。後述のプログラムでも使用する
- **timm**：PyTorch 上で動作する画像認識モデルのライブラリである。多数の **SOTA（State-of-the-Art：最高性能）**モデルが実装されており、転移学習などに活用される
- **NumPy**：数値計算を行うためのライブラリである。多次元配列の操作が可能であり、科学計算の基盤となる
- **Pandas**：データ分析を行うためのライブラリである。表形式データの操作や加工を得意とし、データの前処理に使用される
- **Matplotlib**：グラフ描画ライブラリである。データの可視化に利用され、後述のプログラムでも損失の推移をグラフで表示するために使用する

2.3 プログラム実行による探求の基本手順

プログラム実行による探求は以下のサイクルで行う。

1. **仮説設定**：プログラムの動作について予想を立てる
2. **コード変更**：パラメータや値を変更する
3. **実行**：プログラムを実行して結果を得る
4. **結果観察**：出力やグラフを観察する
5. **考察**：結果から得られた知見を整理し、次の仮説を立てる

このサイクルを繰り返すことで、プログラムの動作原理を理解する。次節で扱う**学習率**の変更は探求の一例であり、様々なパラメータや設定を対象とした探求が可能である。

2.4 学習率の変更と効果

本節では、**学習率**がモデルの学習速度に与える影響を観察する。以下のプログラムを使用して、学習率を変更した際の効果をグラフで確認する。

用語説明

- **損失関数**：予測値と正解値の差を数値化する関数である。この値が小さいほど予測精度が高い
- **勾配降下法**：損失を最小化するために、損失の傾き（勾配）を利用してパラメータを調整する最適化手法である

サンプルプログラム

```
import torch
import matplotlib.pyplot as plt

# 入力データと目標値を定義
x = torch.tensor([1., 2., 3., 4.])
y = torch.tensor([3., 5., 7., 9.])

# 学習する重み 'a' とバイアス 'b' を初期化。requires_grad=True により勾配計算を有効にする
a = torch.tensor(0.5, requires_grad=True)
b = torch.tensor(0.0, requires_grad=True)

# 各反復での損失を記録するためのリスト
losses = []

# 勾配降下法による学習ループ
for _ in range(10):
    # 予測値と目標値の差の二乗平均を損失として計算
    loss = ((a * x + b - y) ** 2).mean()

    # 損失値をリストに追加。.item()でテンソルから Python の数値に変換する
    losses.append(loss.item())

    # 損失の勾配を計算
    loss.backward()

    # 勾配降下法による重み 'a' とバイアス 'b' の更新
```

```
        # .data を直接変更することで、この操作が勾配計算に影響しないようにする
        a.data -= 0.1 * a.grad
        b.data -= 0.1 * b.grad

        # 勾配をゼロにリセット（次の反復のために必要）
        a.grad.zero_()
        b.grad.zero_()

    # 学習後の重み 'a'、バイアス 'b' と最終的な損失値を出力
    print(f"a={a:.2f}, b={b:.2f}, loss={loss:.3f}")

    # 損失の推移をグラフで表示
    plt.plot(losses)
    plt.xlabel('Iteration')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.show()
```

このプログラムは、**線形関数 y=ax+b（一次関数）**の重み a とバイアス b を**勾配降下法**で学習する PyTorch の実装である。入力データ[1,2,3,4]と目標値[3,5,7,9]から、初期重み a=0.5、初期バイアス b=0.0 を **10 回の反復**で最適化し、各ステップの損失値をリストに記録してグラフ表示する。重み a やバイアス b の初期値、**学習率 0.1** を変更することで、収束過程の違いを観察できる。

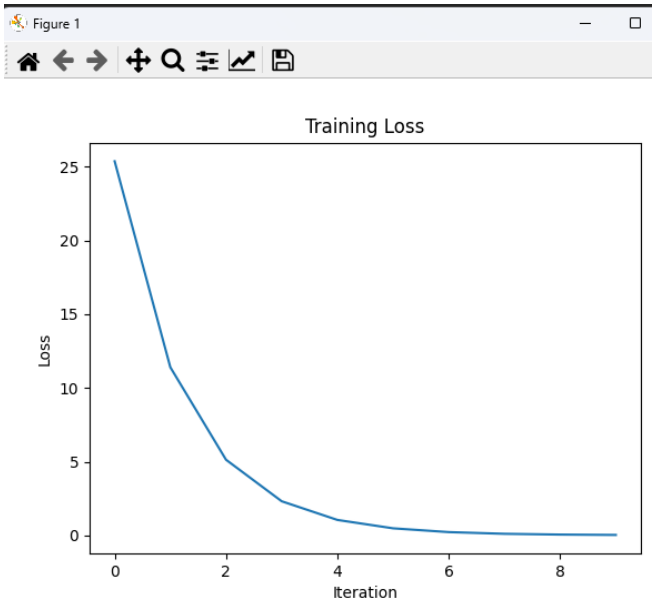
実行結果の確認方法

プログラムが正常に実行されると、以下の結果が得られる。

- コンソールに「a=2.00, b=1.00, loss=0.000」のような出力が表示される
- 損失のグラフが表示され、損失値が反復回数とともに減少する曲線が描かれる

グラフの**横軸は反復回数（0 から 9 まで）**、**縦軸は損失値**を表す。正常な学習では、グラフの線は左上から右下に向かう下降曲線を描く。学習率が適切な場合、滑らかな減少曲線を描いて最終的に**損失がほぼ 0 に収束**する。学習率が大きすぎると振動や発散を示し、小さすぎると変化が緩慢になる。

様々な学習率を試して損失のグラフがどのように変化するかを観察することで、**学習率がモデルの収束に与える影響**を理解できる。



学習率の変更手順

1. ファイルを開き、コードを表示する

2. 以下の行を探す

```
a.data -= 0.1 * a.grad
b.data -= 0.1 * b.grad
```

3. この行の **0.1** を別の数値に変更する

   - 例：**0.01**（学習率を小さくする）

   - 例：**0.5**（学習率を大きくする）

4. コードを変更したら、ファイルを保存する（**Ctrl+S** またはメニューの **File → Save**）

5. 画面上部の**実行ボタン**をクリックしてプログラムを再実行する

期待される効果

学習率を変更すると、表示される損失のグラフに以下のような違いが見られる。

- **学習率を小さくした場合（例：0.01）**：損失はゆっくりと減少する。重み a と b の更新量が小さいため、10 回の反復では損失が十分に下がらない場合がある

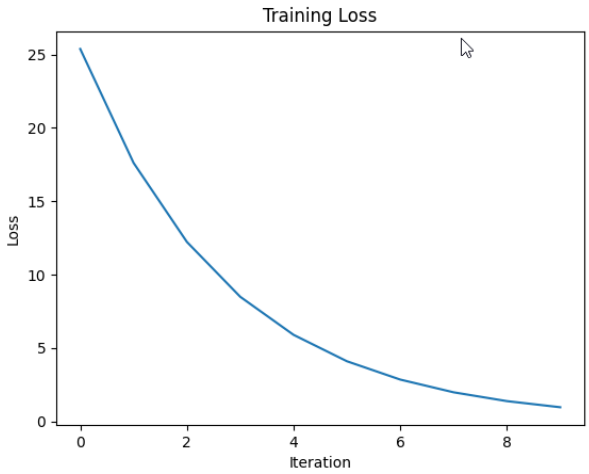


図. 損失の推移グラフの表示例

- **学習率を大きくした場合（例：0.5）**：損失は急激に変化する。適切な範囲内であれば速やかに収束するが、学習率が大きすぎると更新量が過大となり、損失が振動または発散することがある

演習の例

1. 上記のプログラムを実行し、損失のグラフを確認する

2. 学習率を 0.01 に変更して実行し、グラフの変化を観察する

3. 学習率を 0.5 に変更して実行し、グラフの変化を観察する

4. それぞれの実行結果を比較し、**学習率が収束過程に与える影響**を確認する

この演習は一例である。内容に応じた考察、結果の記録、次の仮説設定を行うことが求められる。