

「データ人材育成企画」データ構造化オンライン学習
(Python 初心者向け)

第 6 回

プログラムの作成と実行～まとめ



2023 年 3 月

Smart Solutions 株式会社

1. はじめに

本セミナーも、今回でいよいよ最終回となります。これまで学んできた内容の総集編であり、皆さんに実際にプログラムを作成していただく「実践編」でもあります。内容は、次のとおりです。

- (1) Visual Studio Code の使い方
- (2) 仮想環境の準備
- (3) プログラムの作成
- (4) おわりに

それでは早速、講義をスタートしましょう。

2. Visual Studio Code の使い方

前回の講義の最後でご説明したように、今回の講義は Google Colaboratory ではなく、公式版 Python と Visual Studio Code を使ってプログラムを作成する方法をご紹介します。まずは、Visual Studio Code でプログラムを作成するための準備をします。

ご自身の PC に Python 実行環境と Visual Studio Code をインストールされた方は、ぜひお手元で実際に操作してみてください。

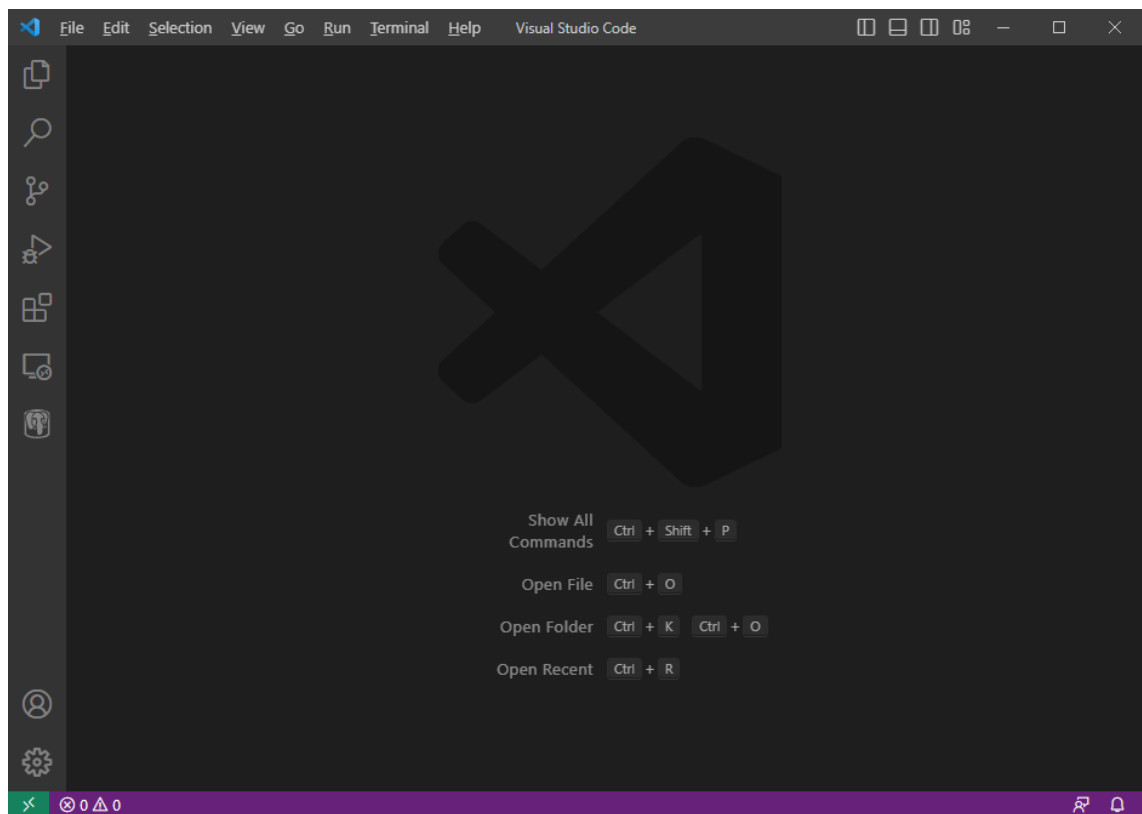
なおこれ以降は、Visual Studio Code のことを略称の「**VSCode**」と記載します。

2.1. 作業用ディレクトリの作成

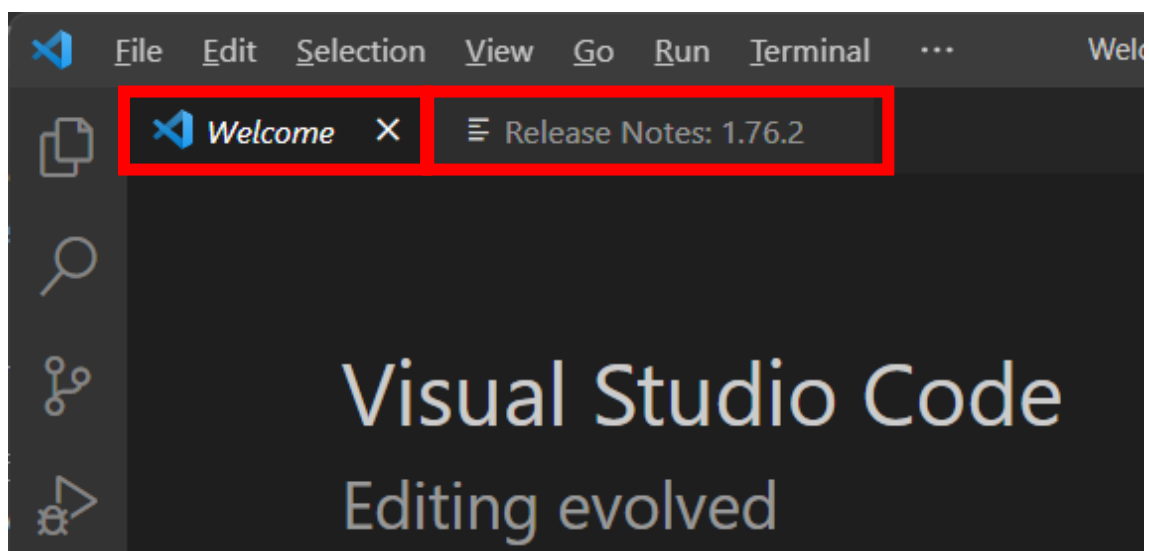
プログラムの作成を始めるにあたり、作業用のディレクトリを作成します。任意のディレクトリを作成してください。

2.2. VSCode の起動

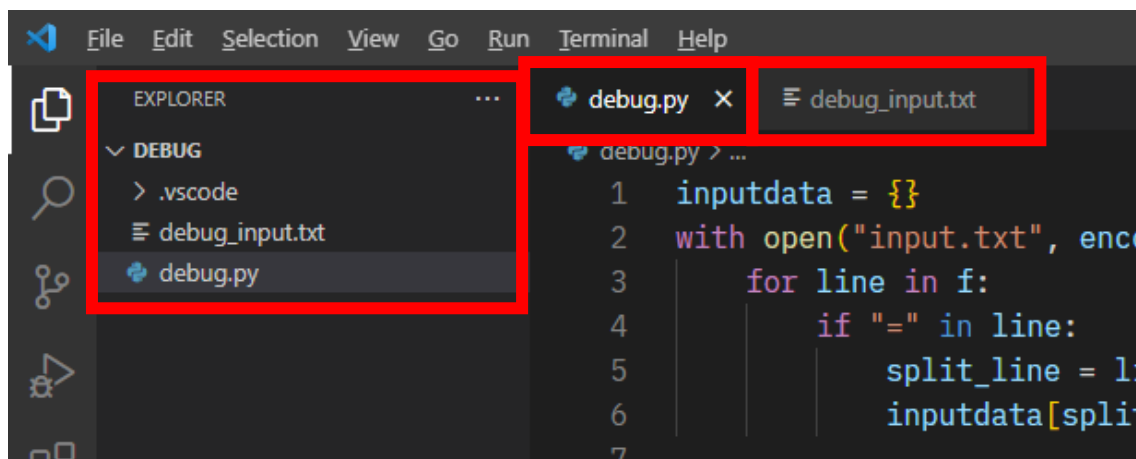
PC にインストールされた VSCode を起動します。初めて起動する場合は、次のような画面になります。



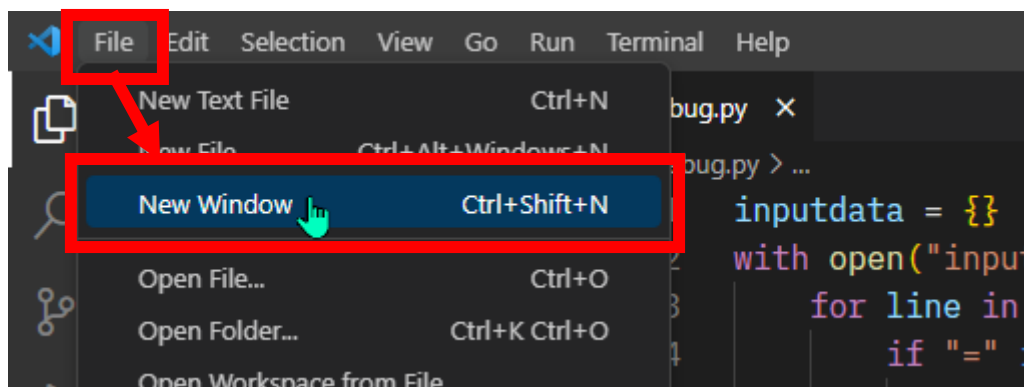
もし「Welcome」や「Release Notes」のタブが開いた場合は、タブ左の「×」をクリックして閉じておきましょう。



また、過去に VSCode を使って作業していた場合、次のように、ソースコードなどのファイルが開かれた状態で起動することがあります。

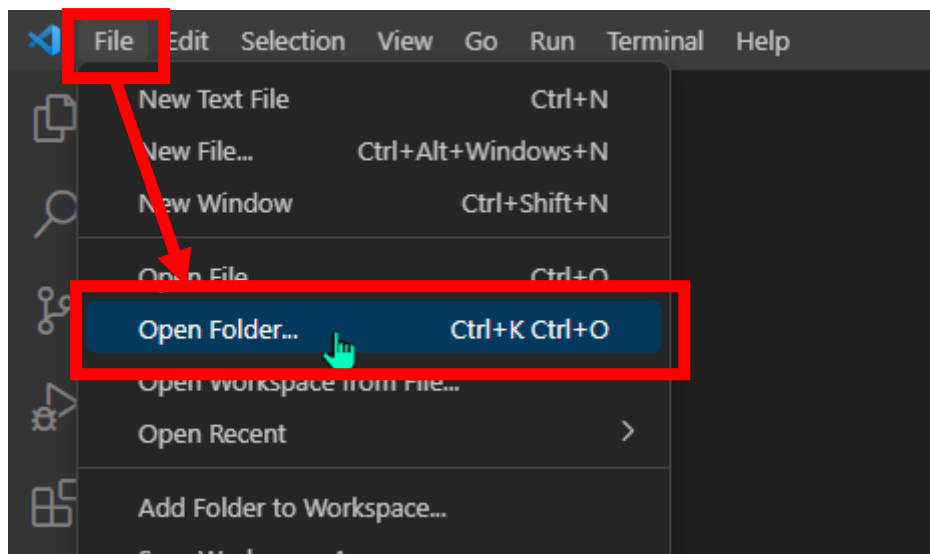


その場合は、VSCode の画面左上のメニューから、[File]→[New Window]を選択して、何も開いていない状態の新規ウインドウを作成してください。

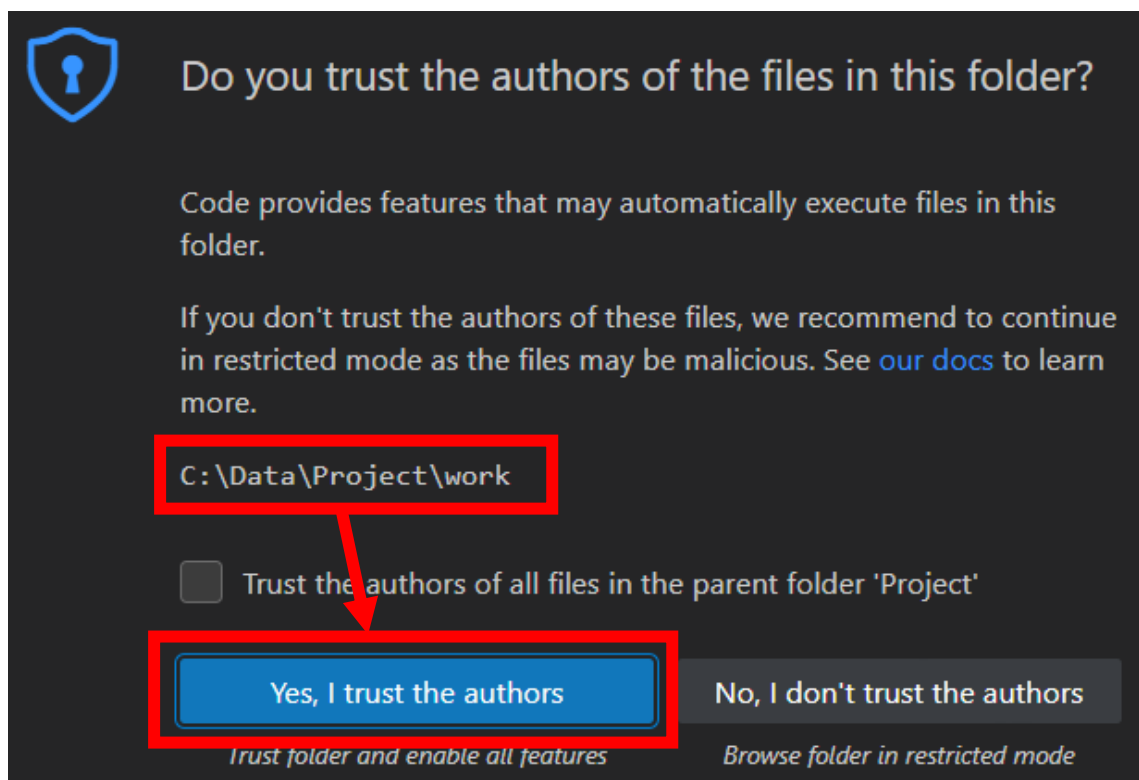


2.3. 作業用ディレクトリのオープン

VSCode の画面左上のメニューから、[File]→[Open Folder]を選択します。フォルダ選択のダイアログが開きますので、あらかじめ作成した作業用ディレクトリを選択してください。



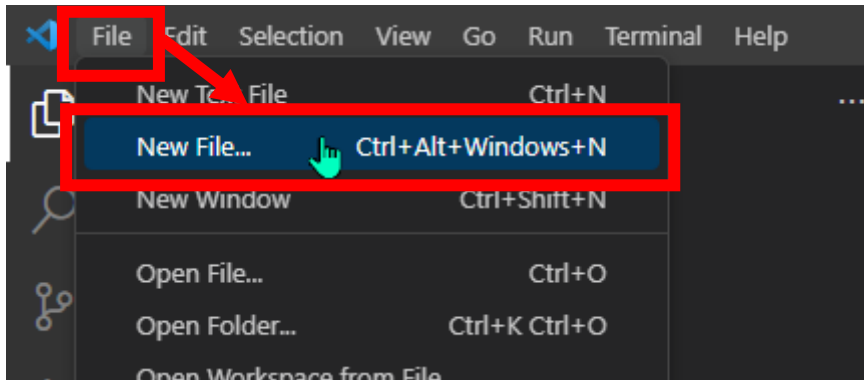
このとき、次のようにフォルダの信頼性を確認するダイアログが出ることがあります。自身で作成した作業用フォルダであることを確認して、「Yes, I trust the authors」をクリックしてください。



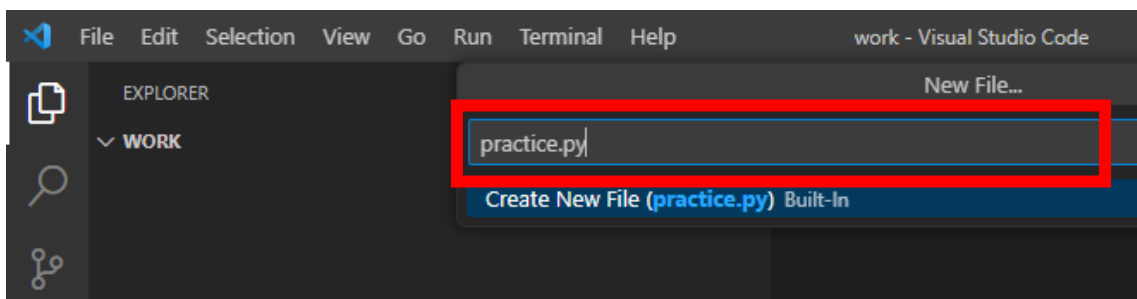
2.4. モジュール(ソースコードのファイル)の作成

第2回の「注意すべき用語」にて、「モジュール」をご紹介しました。モジュールは、Pythonのソースコードを記述する、拡張子「.py」のテキストファイルのことです。これからプログラムの作成を始めるにあたり、モジュールを作成します。

VSCode の画面左上のメニューから、[File]→[New File]を選択します。

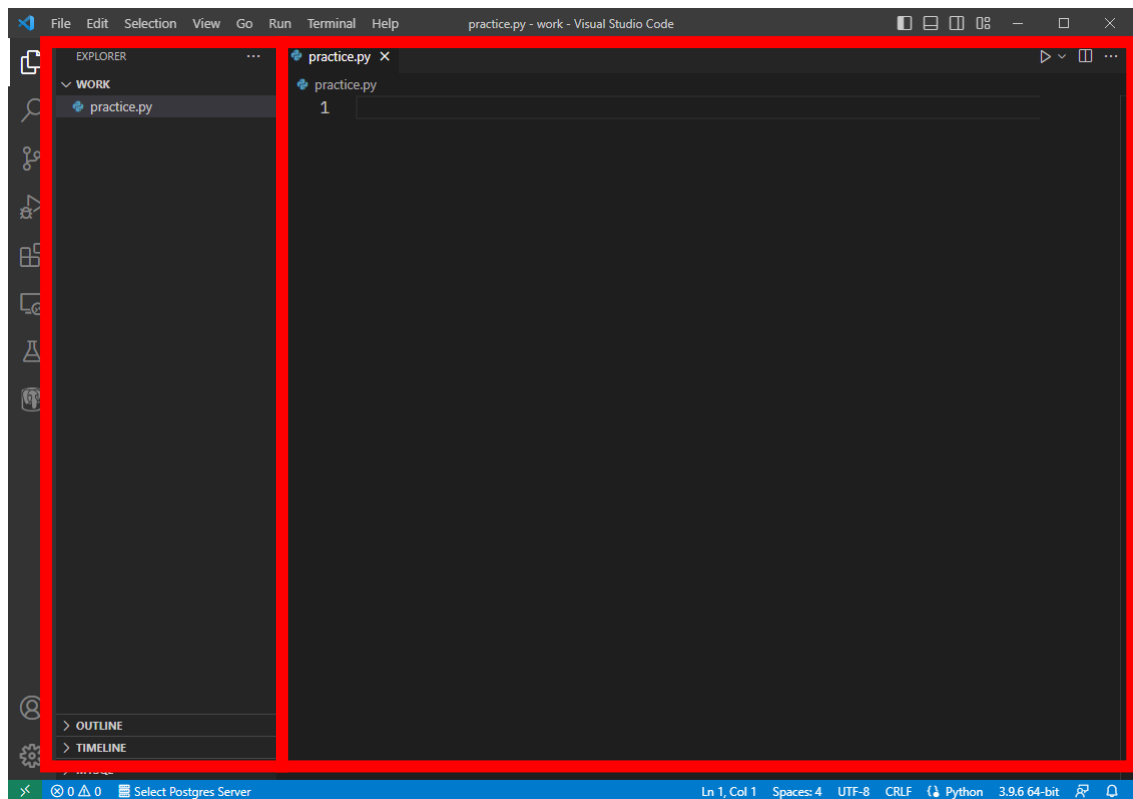


画面中央上部に「Select File Type or Enter File Name...」と記述されたテキストボックスがありますので、そこにモジュール名を入力して Enter キーを押します。



ここでは、「practice.py」とします。ファイル作成のダイアログが開きますので、場所が作業用ディレクトリになっていることを確認して、practice.py という名前でファイルを保存します。

すると、画面左側の「エクスプローラ(EXPLORER)」に作業ディレクトリ内のファイル一覧が、画面右側に先ほど作成した「practice.py」の内容が表示されます。



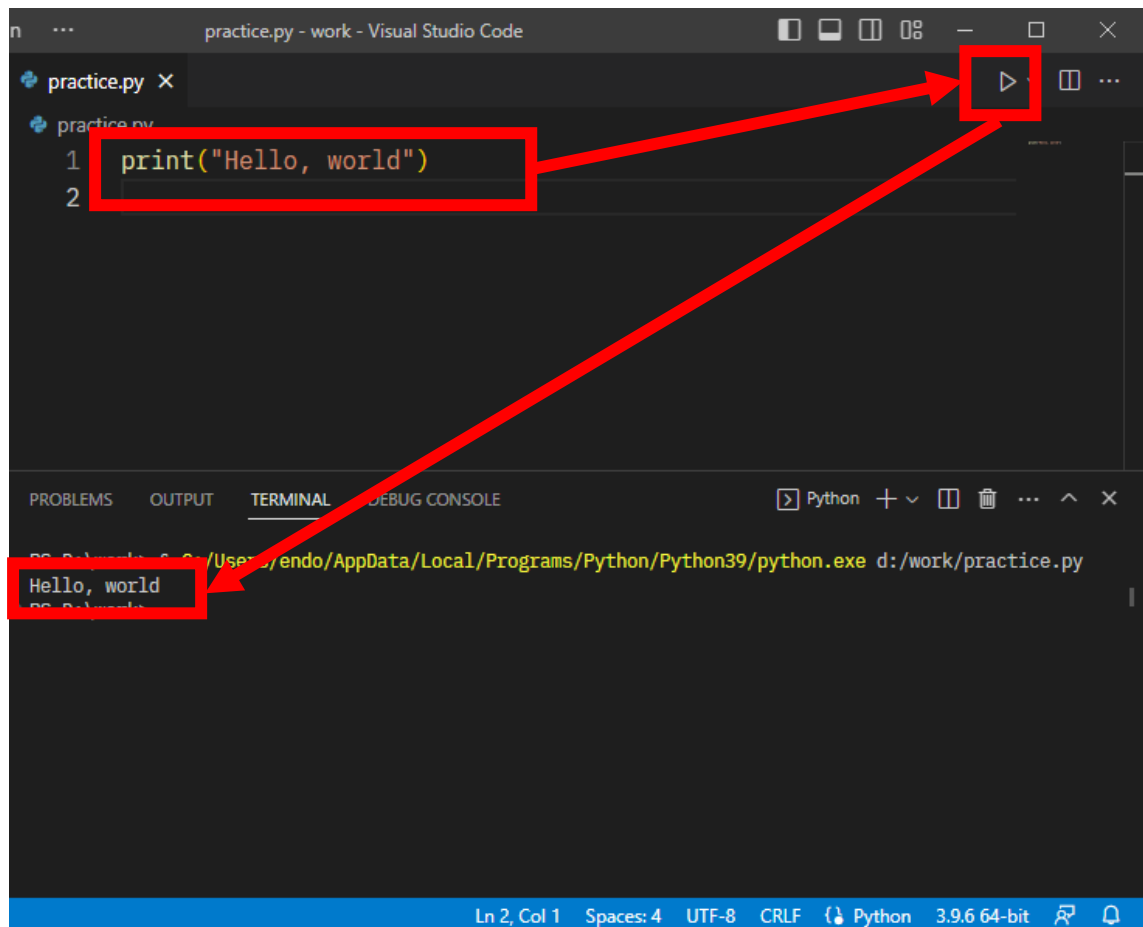
新規作成したファイルなので、右側はまだ空っぽです。この部分はテキストエディタになっており、ソースコードの編集が可能です。

2.5. プログラムを書いて動かしてみよう

とりあえず、簡単なプログラムを動かしてみましよう。例として、画面右側のテキストエリアに下記のプログラムを記述してください。

```
print("Hello, world")
```

記述出来たら、画面右上の三角ボタンを押してプログラムを実行してみてください。画面下部の「ターミナル(TERMINAL)」欄に「Hello, world」と出力されれば、プログラムの実行は成功です。



2.6. プログラムをデバッグしよう

これからもっと本格的なプログラムを作成すると、なかなか思い通りに動かず、原因が分からなくて苦勞することが多くなるでしょう。もしプログラムが想定どおりに動かない場合は、プログラムを動かしながら、原因を突き止めていきます。

プログラムの不具合のことを、プログラミングの世界では「**バグ**」と言います。また、バグを取り除く作業を「**デバッグ**」といいます。

今回は、以下のプログラム `debug.py` と入力ファイル `debug_input.txt` を使って、プログラムのデバッグ方法をご紹介します。次の URL からそれぞれのファイルをダウンロードして、作業用ディレクトリに格納してください。

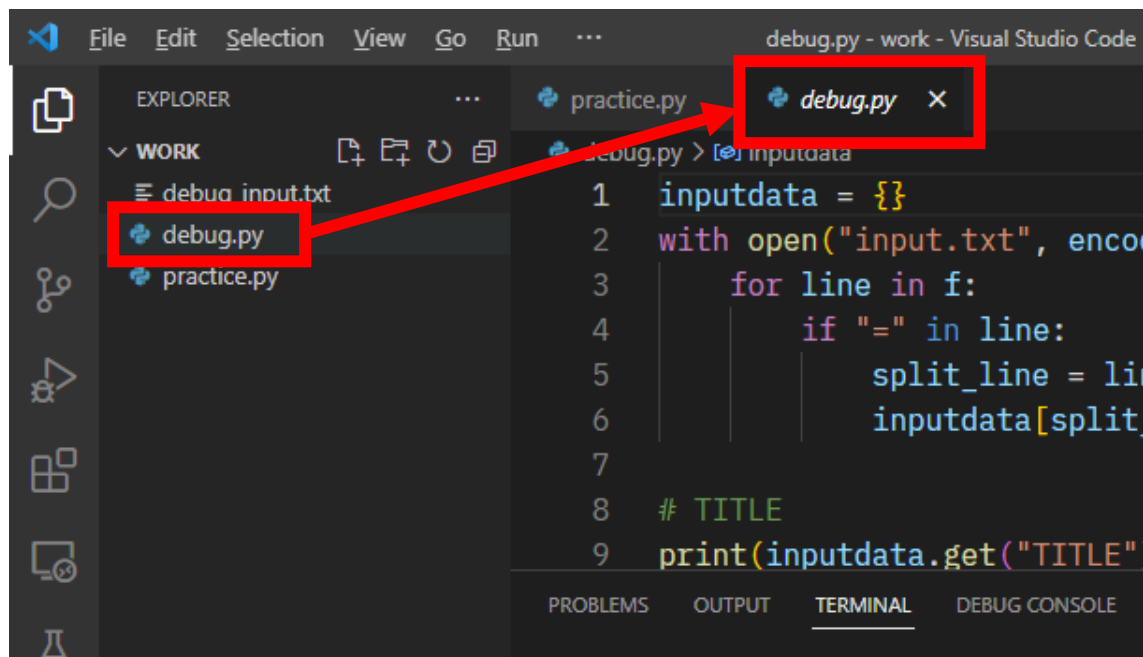
debug.py

https://raw.githubusercontent.com/tendo-sms/python_seminar_2022/main/lecture6/debug.py

debug_input.txt

https://raw.githubusercontent.com/tendo-sms/python_seminar_2022/main/lecture6/debug_input.txt

ファイル一覧に「debug.py」と「debug_input.txt」が追加されます。debug.py をダブルクリックして内容を表示しましょう。



debug_input.txt は、次のようなデータです。各行は「<項目名>= <値>」という形式でデータを定義しています。

debug_input.txt

```
TITLE= MgO SPECTRUM
DATA TYPE= INFRARED SPECTRUM
DATA CLASS= NTUPLES
NUM DIM= 1
ORIGIN= DELTA2_NMR
OWNER = delta
LONGDATE= 2021/10/02 10:29:30
NPOINTS= 1024
$OPERATOR= H.Yamazaki
```

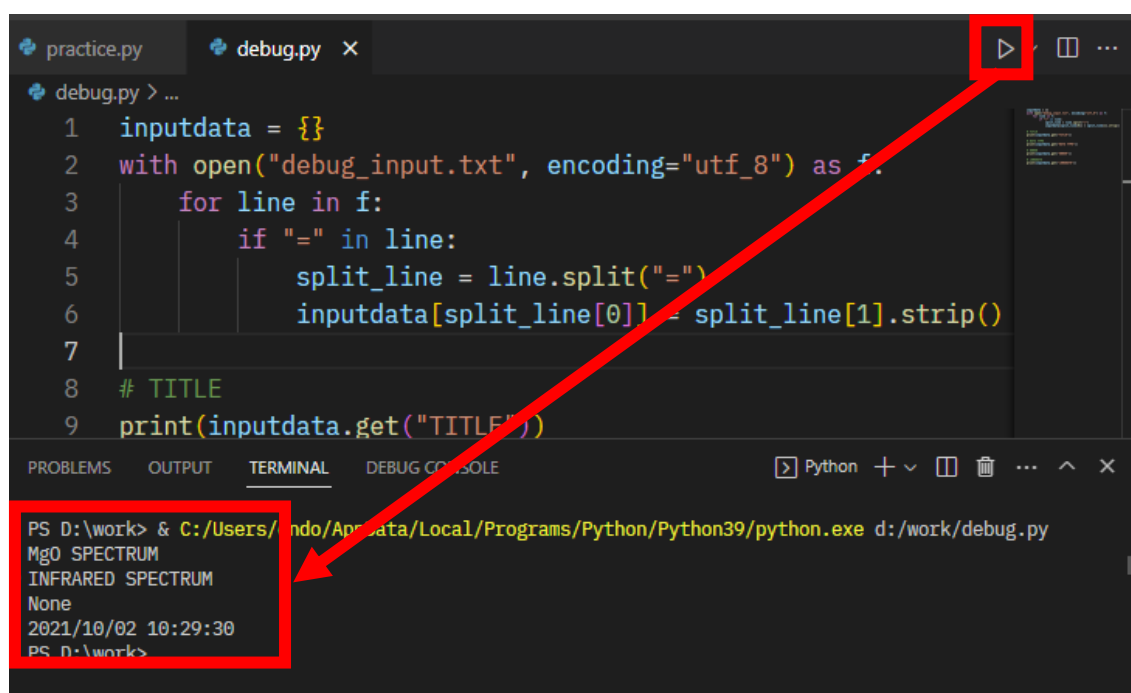
プログラム debug.py は debug_input.txt を読み込んで、次のような辞書を作成します。

{<項目名 1>:<値 1>, <項目名 2>:<値 2>, . . . }

辞書を作成した後、キーが TITLE、DATA TYPE、OWNER、LONGDATE の値をそれぞれ print 関数で出力しています。

2.6.1. サンプルプログラムの問題点

まずは、プログラムをそのまま実行してみてください。



```
debug.py > ...
1 inputdata = {}
2 with open("debug_input.txt", encoding="utf_8") as f:
3     for line in f:
4         if "=" in line:
5             split_line = line.split("=")
6             inputdata[split_line[0]] = split_line[1].strip()
7
8 # TITLE
9 print(inputdata.get("TITLE"))
```

```
PS D:\work> & C:/Users/endo/AppData/Local/Programs/Python/Python39/python.exe d:/work/debug.py
MgO SPECTRUM
INFRARED SPECTRUM
None
2021/10/02 10:29:30
PS D:\work>
```

出力結果

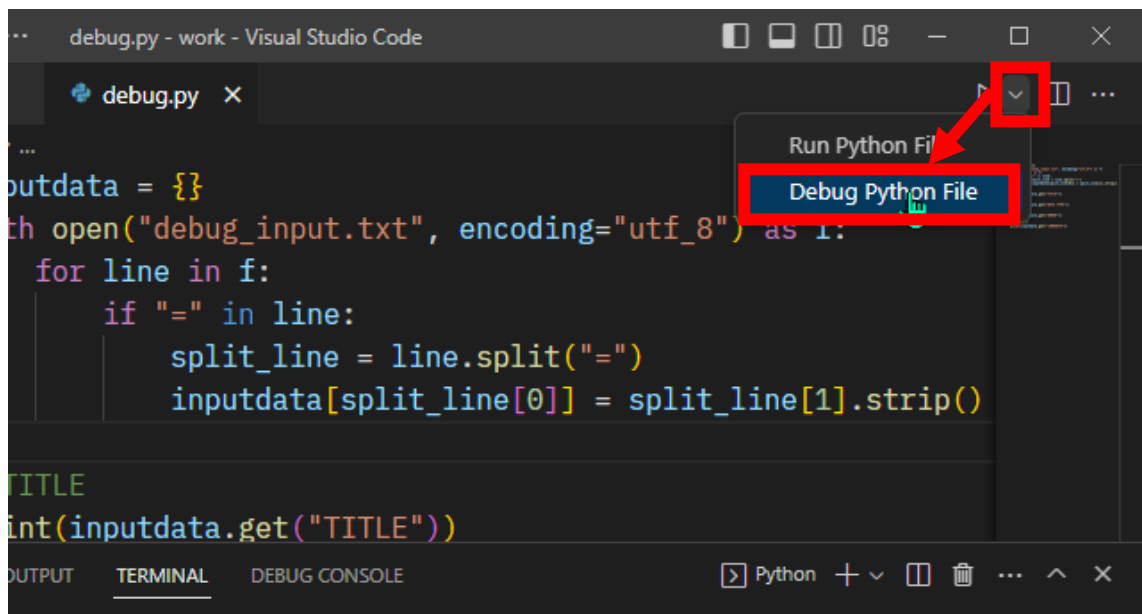
```
MgO SPECTRUM
INFRARED SPECTRUM
None
2021/10/02 10:29:30
```

OWNER の出力結果が、値がないことを示す None になってしまいました。なぜこのような動作になるのか、デバッグで突き止めてみましょう。

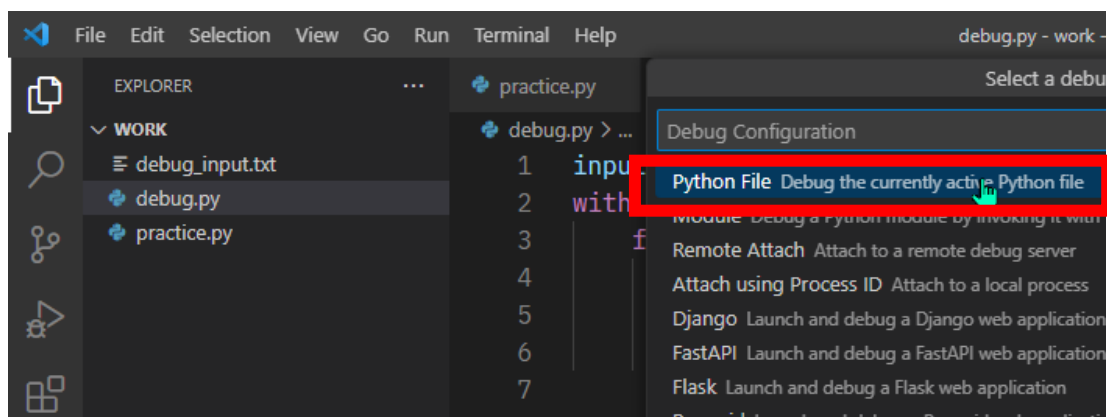
2.6.2. デバッグ実行の方法

デバッグを行うには、通常の実行(Run Python File)ではなく、デバッグ実行(Debug Python File)を使って実行します。次のいずれかの方法で、デバッグ実行を行うことができます。

- キーボードの F5 キーを押す。
- 画面右上の三角ボタンの横にある「v」マークをクリックして、「Debug Python File」を選択する。



なお F5 キーを押してデバッグ実行を開始した場合、画面上部中央に「Debug Configuration」と記述されたテキストボックスが表示されることがあります。そのときは、「Python File」を選択してください。



上記の操作を行うと、デバッグ実行が開始します。ですが、ただ上記の操作を行っただけでは、普通に実行した場合と同じようにプログラムが動くだけです。

デバッグ実行は、次のような機能と組み合わせることで、はじめて原因の調査が可能となります。

- (1) ブレークポイント(BREAKPOINTS)
- (2) 変数(VARIABLES)
- (3) ブレークポイントからの再開

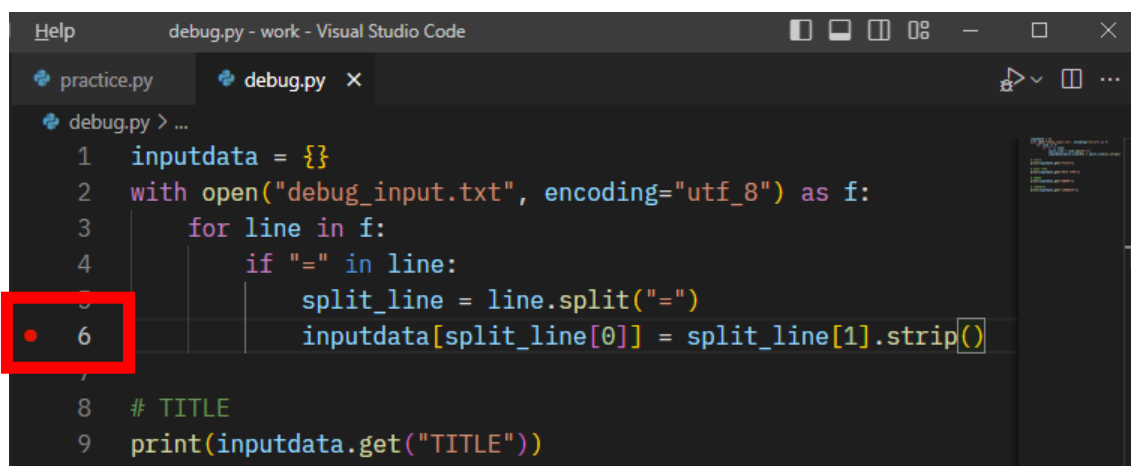
以降では、これらの機能についてご紹介します。

2.6.3. ブレークポイント(BREAKPOINTS)

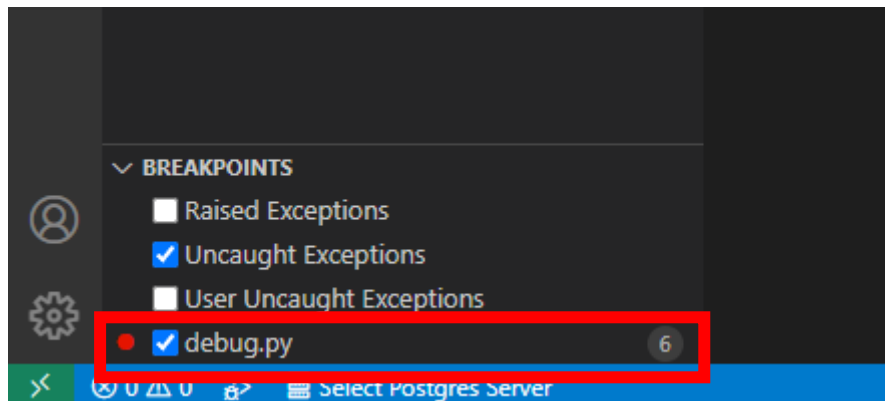
ブレークポイントは、好きな位置でプログラムを一時停止させる機能です。次のいずれかの方法で設定します。ブレークポイントを設定すると、設定した位置の行番号の左側に赤い●が付きます。

- 停止させたい位置にカーソルをセットして、キーボードの F9 キーを押す。もう一度押すと、ブレークポイントが解除される。
- 停止させたい位置の行番号の左側をクリックする。もう一度クリックすると、ブレークポイントが解除される。

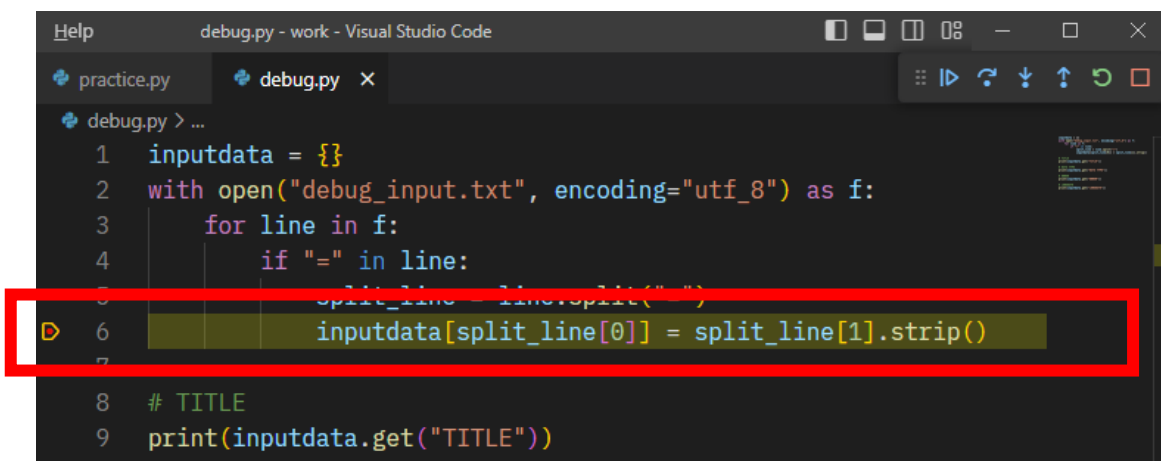
ここでは、6 行目にブレークポイントを設定します。



また、画面左下の「ブレークポイント(BREAKPOINT)」欄にブレークポイントの一覧が表示されます。



ブレークポイントを設定したら、もう一度デバッグ実行をしてみましょう。ブレークポイントでプログラムが一時停止します。一時停止すると、停止中の行がハイライトされます。



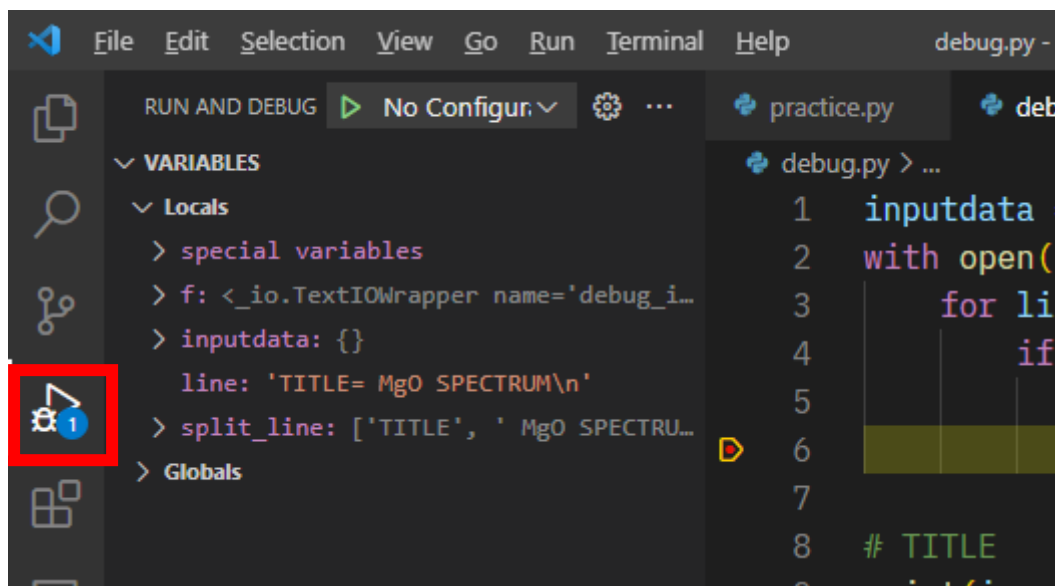
[注意事項]

条件を満たさない if 文の中など、実際に実行されない位置にブレークポイントを設定してもプログラムは停止しませんので、注意してください。

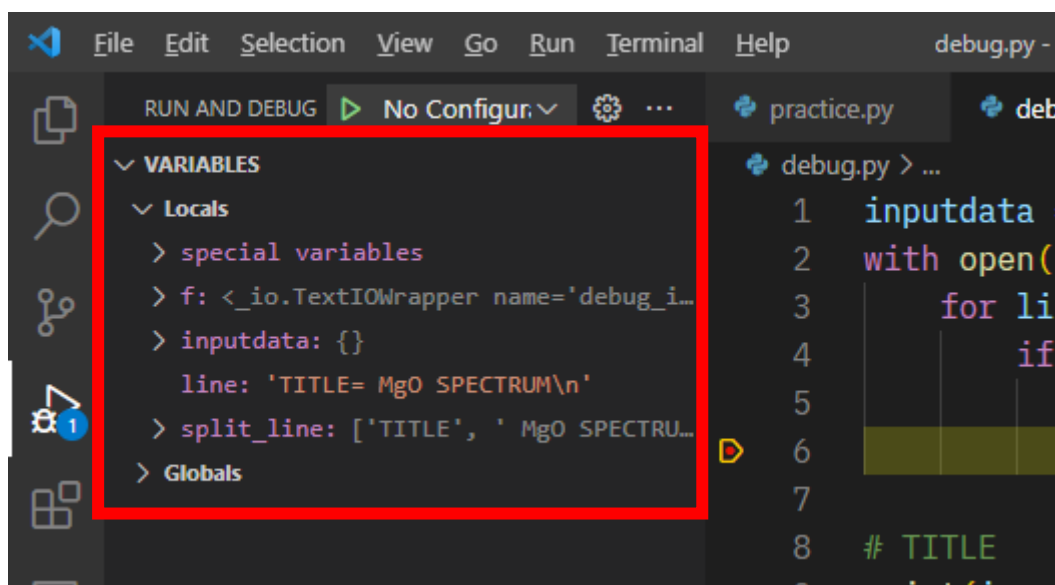
プログラムを一時停止させた状態のまま、次の機能を見てみます。

2.6.4. 変数(VARIABLES)

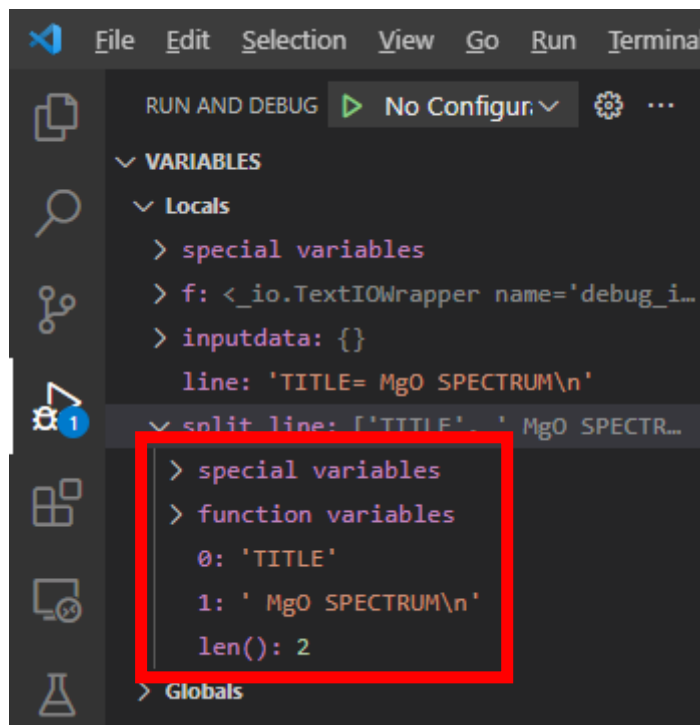
プログラムを一時停止させた状態になると、画面左にあるアイコンのうち、上から4つ目の一番上の Run and Debug アイコン（三角形+虫のマーク）が選択された状態になります。



ここで、画面左上の「変数(VARIABLES)」欄を見てみます。ここには、現在設定されている変数と値の一覧が表示されています。ここを見て、変数の値が期待通りになっているかを確認できます。



split_line の左側の「>」をクリックして、split_line 変数の中身について詳細を表示させてください。



現在は、「=」で区切った左側の”TITLE”をキーに、”MgO SPECTRUM”(split メソッドを呼ぶので前の空白や後ろの改行コード(\n)は削除される)を値にして、辞書を登録しようとしていることが分かります。

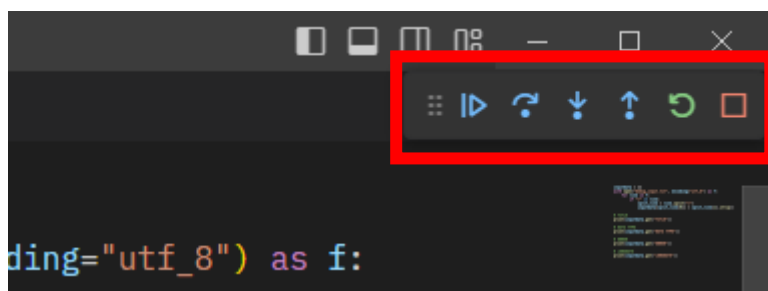
2.6.5. ブレークポイントからの再開

今回問題となっているのはキーが”OWNER”の時に値をきちんと設定しているか、ですので、そこまでプログラムを進めたいところです。ブレークポイントで一時停止している状態から次に進めるには、次の表に示す機能があります。

#	名称	キー入力	機能
1	続行 Continue	F5	次のブレークポイントまで実行して一時停止
2	ステップオーバー Step Over	F10	次の命令を実行して一時停止 現在の行が関数・メソッド呼び出しの場合、関数・メソッドの中には入らない

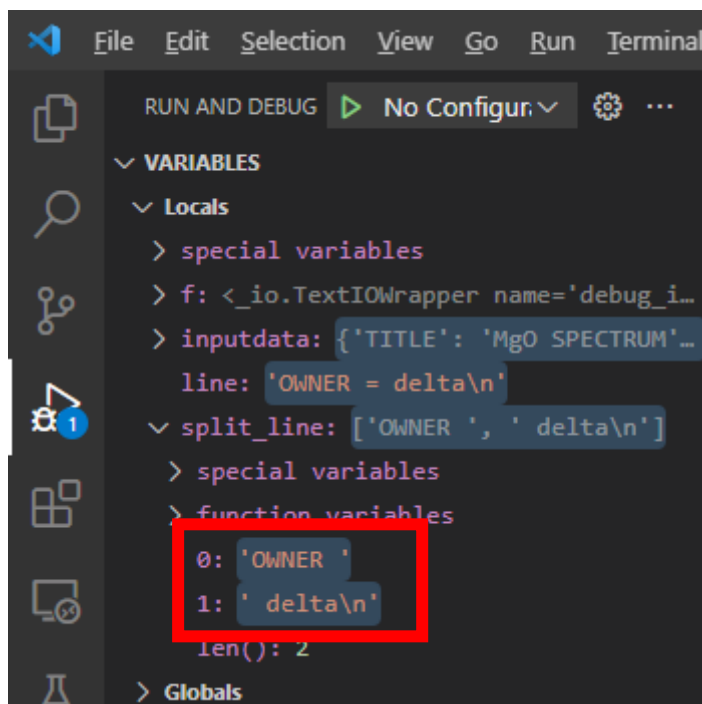
3	ステップイン Step Into	F11	次の命令を実行して一時停止 現在の行が関数・メソッド呼び出しの場合、関数・メソッドの中に入る
4	ステップアウト Step Out	Shift + F11	現在実行中の関数・メソッドが終了するまで実行して一時停止
5	再起動 Restart	Ctrl + Shift + F5	もう一度最初からプログラムを実行
6	停止 Stop	Shift + F5	プログラムの実行を停止

これらはキー入力に加えて、画面のデバッグメニューからも行えます。左から「続行」「ステップオーバー」「ステップイン」「ステップアウト」「リスタート」「停止」となります。(マウスカーソルを合わせると機能名が表示されます。)



2.6.6. バグの正体

今回は、"OWNER"をキーとして辞書を登録するところまで、「続行(Continue)」を繰り返します。何回か「続行(Continue)」を行うと、「変数(VARIABLES)」欄が次のようになります。



split_line の詳細の中で、「0:」の部分をよく見てください。この値がキーとなりますが、「OWNER」ではなく、「OWNER 」となっており、末尾に空白が入ってしまっています。

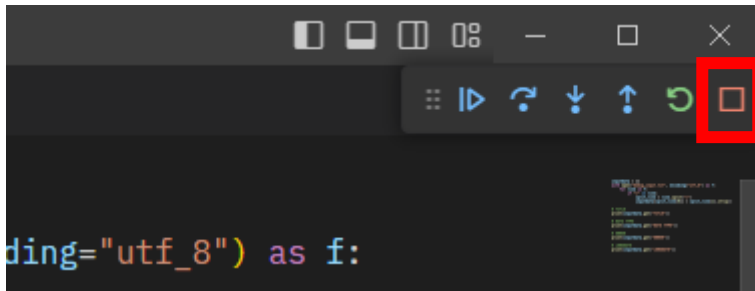
ここで debug_info.txt をよく見てみると、OWNER の行は次のようになっています。「OWNER」と「=」の間に空白が入っていますね。

```
OWNER = delta
```

「=」の左側をキーとしたため、キーが「OWNER」ではなく「OWNER<空白>」となってしまいました。これが、バグの正体です。

2.6.7. デバッグの停止

バグの正体が分かりましたので、デバッグを停止します。キーボードで Shift + F5 を押すか、デバッグメニューの口ボタンを押してデバッグを停止します。



2.6.8. プログラムの修正と再実行

「=」の右側だけではなく、左側にも空白が入る可能性がありますので、キー側にも strip 関数を入れるようにしましょう。次のように、プログラムを変更します。

変更前

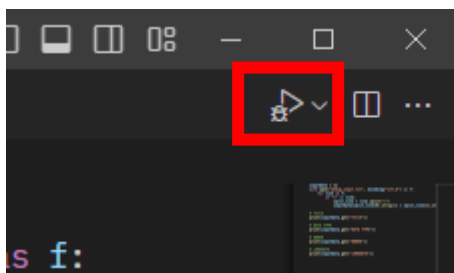
```
inputdata[split_line[0]] = split_line[1].strip()
```

変更後

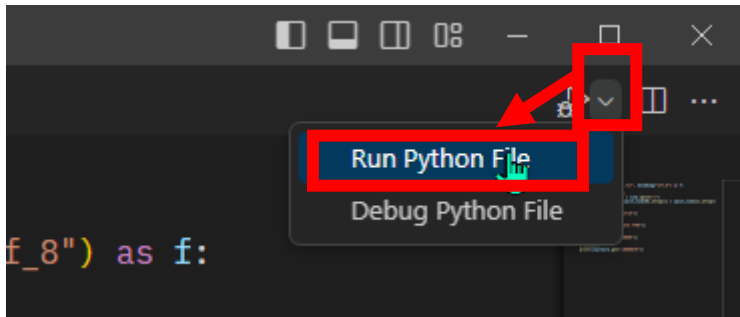
```
inputdata[split_line[0].strip()] = split_line[1].strip()
```

変更したら、もう一度プログラムを実行してみましょう。今度は、OWNER の値が正しく表示されます。

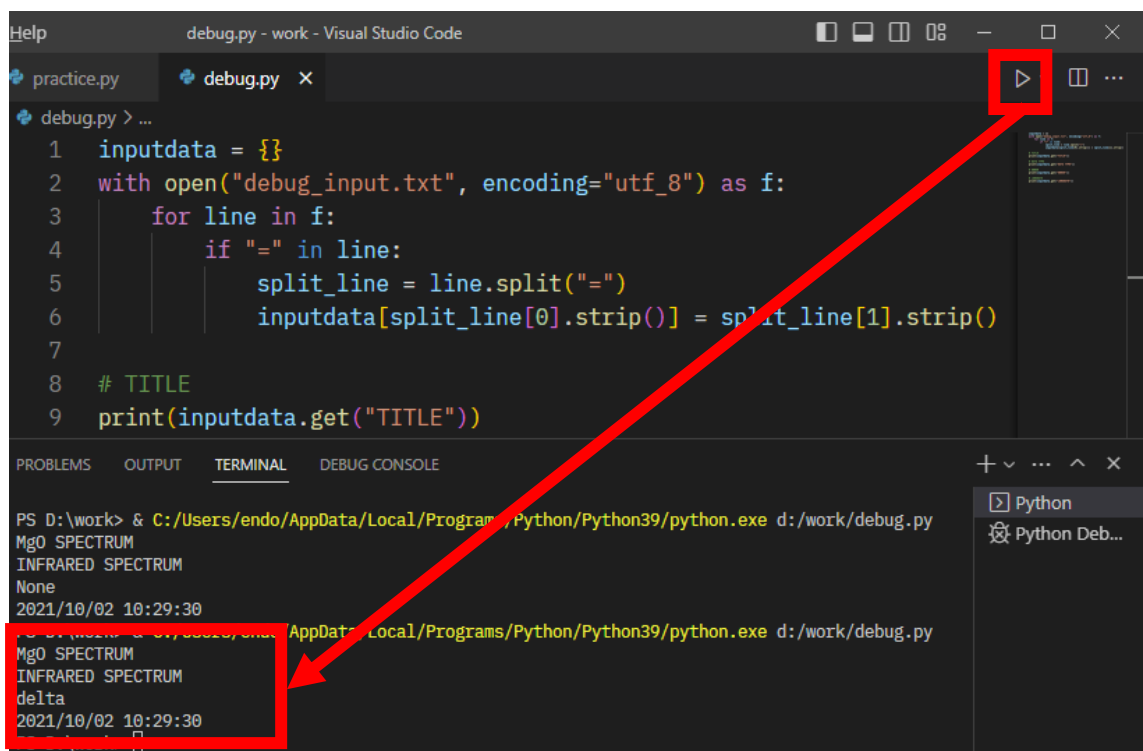
なお、『画面右上の三角ボタンの横にある「v」マークをクリックして、「Debug Python File」を選択する。』でデバッグ実行を行った場合、画面右上の三角ボタンに虫のマークがつけます。



この状態でボタンを押すと、デバッグ実行となります。通常実行する場合は、横にある「v」マークをクリックして、「Run Python File」を選択してください。



今度は、“OWNER”の値が正しく表示されます。



出力結果

```
MgO SPECTRUM
INFRARED SPECTRUM
delta
2021/10/02 10:29:30
```

今回ご紹介したデバッグの機能はごく一部ですが、本格的なプログラムを作るときは、このような便利な機能をフルに活用して、よくデバッグをしましょう。

3. 仮想環境の準備

VSCoide の基本的な操作方法を学んだところで、いよいよ今回の課題となるプログラムの作成に進んでいきます。ですがその前に、Python でのプログラム開発において非常に重要な「**仮想環境**」についてご説明いたします。

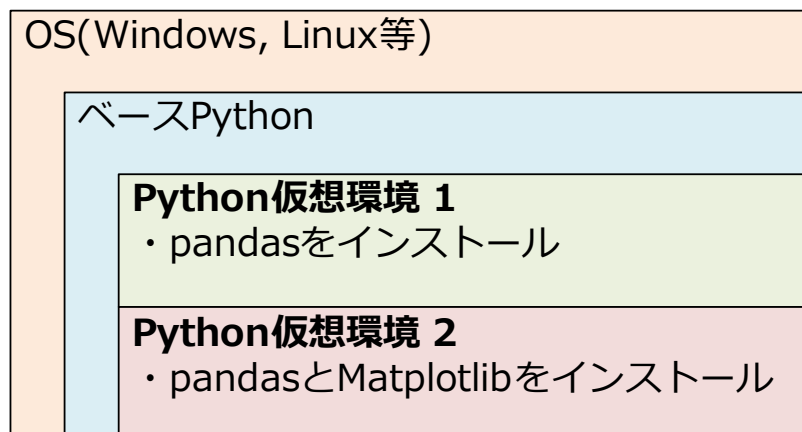
3.1. 仮想環境とは

プログラム作成の環境が整ったところで、早速作成を開始・・・するまえに、Python の「仮想環境」についてご説明します。

Python で本格的なプログラム開発を行う際、プログラムごとに専用の「仮想環境」を作成して、仮想環境上でプログラムを開発することが推奨されます。

「仮想環境」というと、Windows 上で仮想的に Linux を動かしたり、Mac 上で仮想的に Windows を動かしたり・・・といった、OS の仮想化をイメージされるかもしれませんが。しかし、Python の仮想環境はちょっと違います。

Python の仮想環境は、システムの Python 環境(ベース Python と呼ばれます)の上に、独立した複数の環境を作ることができます。



環境を分けることによって、「環境 1 には pandas だけをインストールする」「環境 2 には pandas と Matplotlib をインストールする」といったことができます。

また、同じパッケージをインストールするとしても、仮想環境ごとに異なるバージョンのパッケージをインストールすることもできます。この点は特に重要で、あるプログラムを動かすためにパッケージをバージョンアップさせたら、今度はこれまで動いていた Python プログラムや他のパッケージが動かなくなってしまった、といった事態が容易に起こりえます。それを防ぐためにも、プログラムごとに仮想環境を作成して、個々のプログラム専用のパッケージをインストールすることが有効です。

3.2. 仮想環境の作成

仮想環境を作成する手順をご紹介します。この手順は、公式版 Python での手順となります。Anaconda など他の Python 実行環境では手順が異なりますので、ご注意ください。

端末(Windows のコマンドプロンプトまたは PowerShell、Linux のログインシェル、など)で操作を行います。

cd コマンド等で、カレントディレクトリを作業用ディレクトリに設定してください。

```
cd 作業用ディレクトリ
```

仮想環境の作成には、Python が標準で提供する「**venv**」というモジュールを使用します。次のコマンドを実行してください。「**仮想環境名**」は任意です。

Windows の場合

```
py -3.11 -m venv 仮想環境名
```

「3.11」の部分は、各自でインストールされている Python のバージョンに合わせてください。配布した手順書通りに作業されたのであれば、「3.11」で問題ありません。

Linux の場合

```
python3 -m venv 仮想環境名
```

ここまでの操作に成功すると、作業用ディレクトリに **仮想環境名** を名称とするディレクトリが作成されます。

3.3. 仮想環境の開始

次のコマンドを実行すると仮想環境に入ることができます。

Windows の場合

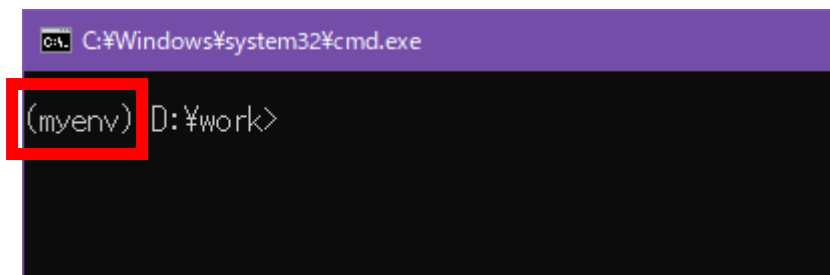
```
仮想環境名¥Scripts¥activate
```

Linux の場合

```
. 仮想環境名/bin/activate
```

Linux では、最初に「.」を忘れずに入れるようにしてください。

端末上でプロンプト文字列に「(仮想環境名)」と表示されていれば、仮想環境に入ることができています。

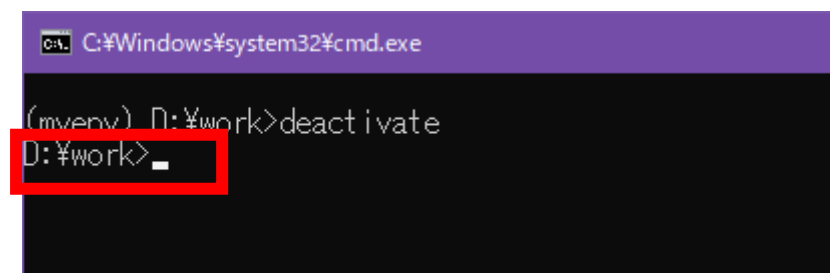


この状態で Python を使った操作を行うと、それは「仮想環境」という名称の仮想環境上での操作ということになります。

なお、仮想環境を抜けるときは次のとおり実行します。Windows、Linux とも共通です。

```
deactivate
```

仮想環境を抜けると、プロンプト文字列に「(仮想環境名)」と表示されなくなります。



3.4. 仮想環境でのパッケージインストール

仮想環境に入った状態で、今回のプログラムで使用する 2 つのパッケージ、pandas と Matplotlib をインストールしましょう。Google Colaboratory ではこれらのパッケージがインストール済みでしたが、公式版 Python では自分でインストールする必要があります。

パッケージのインストールには、第 3 回でご紹介した「**pip コマンド**」を使用します。まずは実際にパッケージをインストールする前に、この「pip コマンド」を最新の状態にしておくことをお勧めします。

仮想環境に入った状態で次のコマンドを実行すると、オンラインで pip コマンドのバージョンをチェックして、最新の状態にすることができます。

Windows の場合

```
python -m pip install --upgrade pip
```

Linux の場合

```
pip install --upgrade pip
```

pip コマンドを最新の状態にしたら、pandas と Matplotlib をインストールしましょう。

pandas のインストール

```
pip install pandas
```

Matplotlib のインストール

```
pip install matplotlib
```

これで、**今回作成した仮想環境でのみ**、pandas と Matplotlib が使用できるようになりました。

3.5. VSCode での仮想環境設定

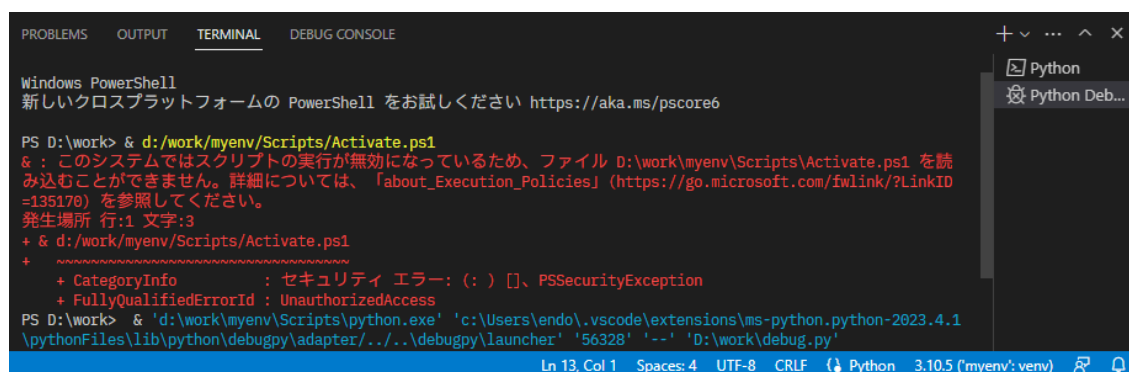
これまで行ってきた VSCode によるプログラム実行およびデバッグは、仮想環境を使用せずに、ベース Python を使って行ってきました。そのため、同じやり方では仮想環境にインストールした pandas や Matplotlib を利用できません。

このあとのプログラム開発ではこれらのパッケージを利用しますので、VSCode で仮想環境を使用するように設定します。

キーボードの Ctrl + Shift + P キーを押すと、画面上部中央にテキストボックスが表示されます。ここで、「Python: Select Interpreter」を選択します。すると、利用可能な Python の環境の一覧が表示されますので、「Python 3.11.2 ('venv': 仮想環境名)」を選択します。これで、VSCode でのプログラム実行が仮想環境で実行されるようになりました。

【注意事項】

事前に展開させていただいたインストール手順書にてご説明した「PowerShell 実行ポリシーの設定変更」を行っていないと、次のような PowerShell のエラーが出力されて仮想環境に入れない場合があります。



```
Windows PowerShell
新しいクロスプラットフォームの PowerShell をお試しください https://aka.ms/pscore6

PS D:\work> & d:/work/myenv/Scripts/Activate.ps1
& : このシステムではスクリプトの実行が無効になっているため、ファイル D:\work\myenv\Scripts\Activate.ps1 を読み込むことができません。詳細については、「about_Execution_Policies」(https://go.microsoft.com/fwlink/?LinkID=135170) を参照してください。
発生場所 行:1 文字:3
+ & d:/work/myenv/Scripts/Activate.ps1
+ ~~~~~
+ CategoryInfo          : セキュリティ エラー: (:) [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess

PS D:\work> & 'd:\work\myenv\Scripts\python.exe' 'c:\Users\endo\.vscode\extensions\ms-python.python-2023.4.1\pythonFiles\lib\python\debugpy\adapter\..\..\debugpy\launcher' '56328' '--' 'D:\work\debug.py'
```

4. プログラムの作成

ここまで長い説明となりましたが、ようやく、最終課題のプログラム作成にチャレンジしましょう。

4.1. 今回作成するプログラム

今回作成するプログラムは、次の動作を行うものです。

- (1) スペクトルデータ(CSV ファイル)を読み込む
- (2) 読み込んだスペクトルデータの空行を削除する
- (3) グラフ情報ファイル(テキストファイル)を読み込む
- (4) スペクトルデータから 2 本の折れ線グラフを作成する
- (5) グラフ情報ファイルのデータを用いてグラフを装飾する
- (6) 作成・装飾したグラフを画像ファイル(DPI は 100、PNG ファイル)に保存する

スペクトルデータ(spectrum.csv)

https://raw.githubusercontent.com/tendo-sms/python_seminar_2022/main/lecture6/spectrum.csv

グラフ情報ファイル(graph_info.txt)

https://raw.githubusercontent.com/tendo-sms/python_seminar_2022/main/lecture6/graph_info.txt

スペクトルデータ(spectrum.csv)は、次のような内容です。ただし、所々に空行(すべてのカラムに値がない行)があるため、そのままグラフにすると折れ線が歯抜けになってしまいます。上記(2)のとおり、空行を削除するようにしてください。

Wavelength	Intensity1	Intensity2
300	409	298
310	447	349
320	383	393
330	563	480

グラフは、横軸を Wavelength、縦軸を Intensity1 および Intensity2 として、2 本の折れ線グラフを重ねて表示してください。凡例も表示してください。

また、グラフのタイトル、軸ラベル、および凡例のラベルを、グラフ情報ファイル(graph_info.txt)から取得してください。グラフ情報ファイルは、次のようなデータです。

```
[Title]
TITLE= Spectrum Graph

[Labels]
XLABEL=Wavelength
YLABEL=Intensity

[Legend labels]
LINE1 = Int1
LINE2 = Int2
```

グラフタイトルは TITLE、軸ラベルは XLABEL および YLABEL、凡例のラベルは LINE1 および LINE2 の値を使用してください。

まさに、これまで学んできたことの集大成という感じにしています。新しい内容はありませんので、卒業試験のつもりでチャレンジしてみましょう。

4.2. プログラム作成

事前にいくつか、プログラム作成のヒントをお伝えします。

- **機能ごとに関数を作りましょう。**ソースコードが分かりやすくなります。(1)～(6)をどのように関数として分けるのが適切か、考えてみてください。
- 基本的には、第 1 回～第 5 回までの講義でご紹介した機能を使って、すべて作成することができます。ですが、今までの講義ではご紹介していない、とあるモジュールを import することで、簡単に実現できる部分があります。「この処理って、すでに誰かがモジュールを公開しているのでは？」とピンと来た方は、Web 等でちょっと調べてみてください。

それでは実際に、今回の題目となるプログラムを作成してみてください。「2.4 モジュール(ソースコードのファイル)の作成」の手順で新たにモジュールを作成して、そこにプログラムを書いていきましょう。

ある程度時間を置きながら、追加のヒントを出していきます。

また、これまでの講義資料の中で、今回の課題に役立つ内容をまとめた資料を下記に掲載します(3/20 にメールで展開させていただいたものと同一)。

https://raw.githubusercontent.com/tendo-sms/python_seminar_2022/main/lecture6/review.pdf

4.3. 解答例

プログラムは完成しましたか？最後に、解答例をご紹介します。(解答例のソースコードは、セミナー終了後に皆様へ展開します。)

【ご参考】環境の削除

仮想環境を削除したい場合は、作業用ディレクトリの下に作成された「仮想環境名」の名称のディレクトリを丸ごと削除します。

5. おわりに

以上で、『「データ人材育成企画」データ構造化オンライン学習（Python 初心者向け）』のセミナーはすべて終了となります。

今回のセミナーは、一般の入門講座のように基礎をみっちり学ぶのではなく、データ構造化プログラムで活用できる機能を中心に、限られた時間の中でプログラミングの楽しさ・便利さを知ってもらうことを目的としておりました。セミナーを通して、「もっとプログラムを作りたいな」「もっと色々な機能を知りたいな」などなど、感じていただけたら、ぜひ、ご自身でさらに勉強してみてください。

「この作業、めんどくさいな」「このデータ、いったいどうやって読み取るの??」といった場面がありましたら、まずは「Python を使えばラクかも!？」と考えてみてください。本セミナーにて学んだことで、皆様の研究・お仕事を何倍にも効率アップさせていただければ幸いです。

皆様、大変お忙しい中、半年間・全 6 回に渡るセミナーの受講、大変お疲れ様でした。

6. アンケートのお願い

最後に、以下の URL からアンケートのご協力をお願いいたします。

<https://docs.google.com/forms/d/e/1FAIpQLSf0-uz1VoC1dS9g0whPsJK5nBePuV84XDGIgoxpKEGW2zsDjA/viewform?usp=sharing>

7. お問い合わせ先について

本書の内容についてご不明な点などございましたら、次の問い合わせ先まで、メールにてお問い合わせください。

Smart Solutions 株式会社（セミナーインストラクター）

担当：遠藤

メールアドレス： workshop_shokyu@smt-sol.jp