Visual Studio Codeの利用

初期設定、実行、デバッグ実行入門

＊デバッグ実行はおまけ＊

Visual Studio Codeとは何か？

- 開発環境の一つ。
- テキストエディタより高機能。
  - エディタ（コード編集）
  - 名称補完機能
  - ヘルプ機能
  - 実行
  - デバッグ
  - （実行後のメモリを保持したままインタプリタ継続実行可能。）

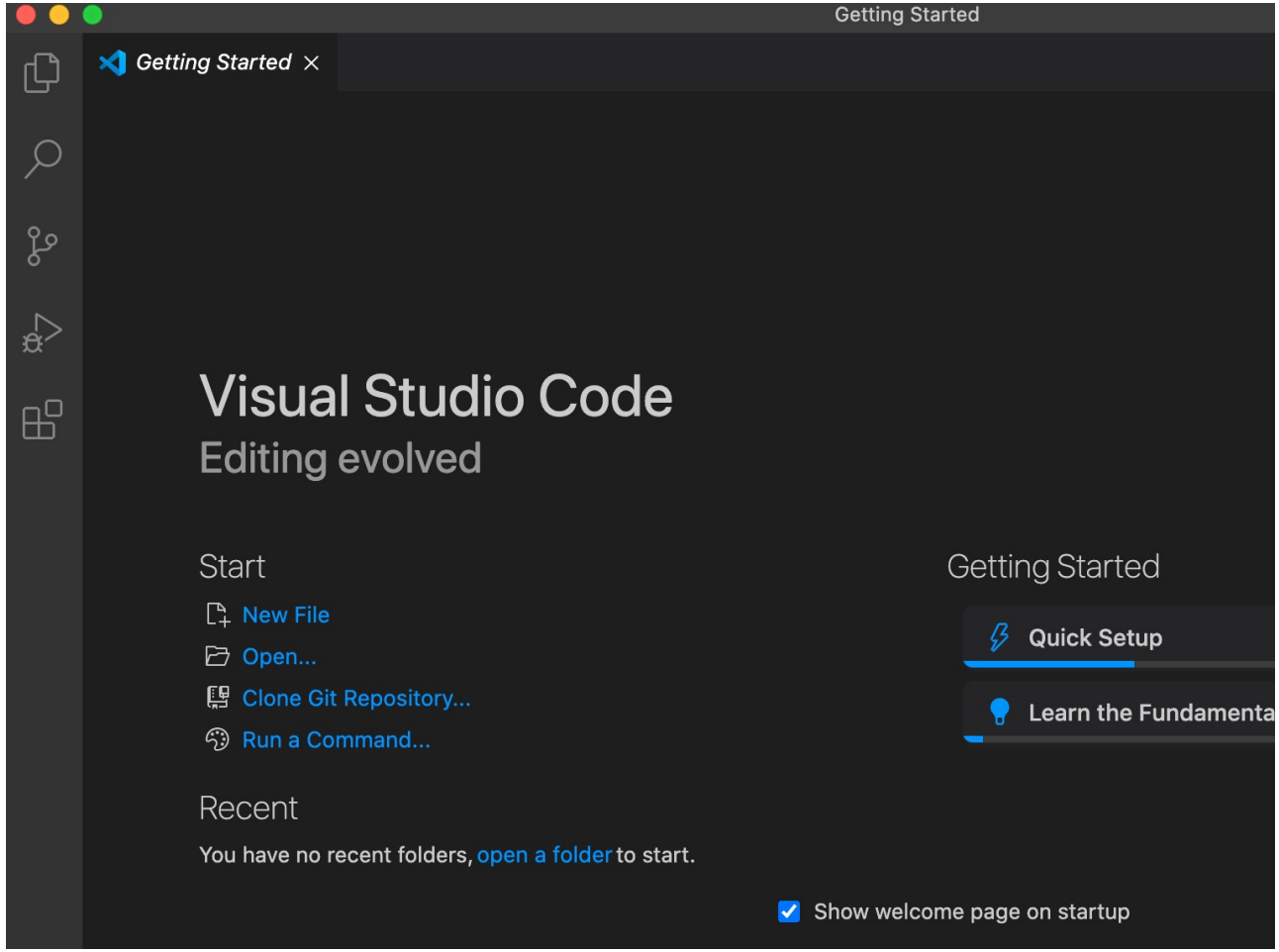
- これまで使わなかった理由
  - ターミナルを使う演習を兼ねていた。
  - 「開発環境がないと実行できない」という勘違いを防ぐため。
    - VS Codeでの実行も、基本的にはPythonインタプリタを使う。

インストール

- https://azure.microsoft.com/ja-jp/products/visual-studio-code/
  - 上記公式サイトからmacOS版をダウンロード。
    - 2021年4月時点では Intel版 を選ぼう。他はまだ不具合あります。
  - 「Visual Studio Code.app」がダウンロードされるので、これをApplicationsフォルダにドラッグ＆ドロップで設置。これでインストール終了。
  - 途中でセキュリティ指摘されて進まなくなる場合には、
    - 左上のリンゴマークをクリックして「システム環境設定」を選択。
    - 「セキュリティとプライバシー」を選び、一般タブを選択。
    - ここでアプリケーションへの実行許可を与えよう。
  - インストール後はDockに登録しておくと便利。

起動

- アプリケーションフォルダもしくはDockから、Visual Studio Code.app を起動。
- 右図のように Getting Started になるはず。
- ここから New File を選択して新規ファイルを作成。
- print(1) と書いて ~/prog1/test.py に保存。



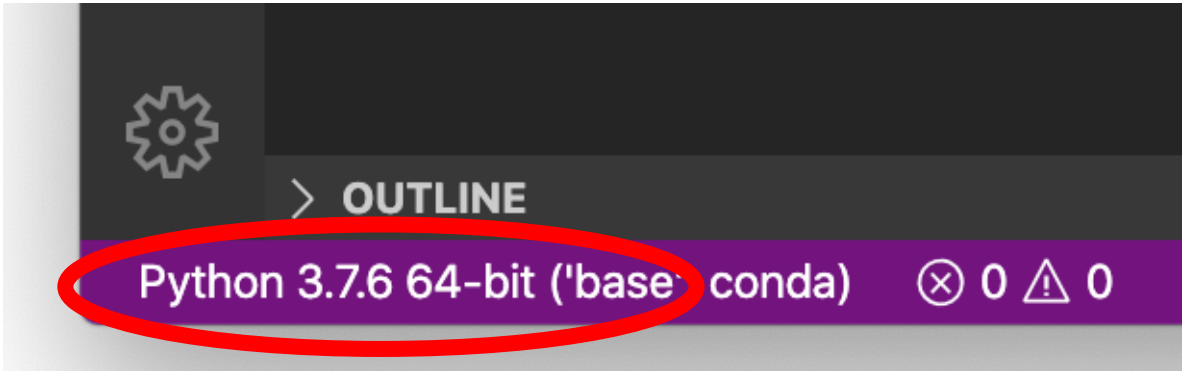
(必要ないかもしれない手順)

- Python Extension
  - Python Extensionのインストールを求められる場合、指示通りに進めてインストールしてください。
  - 求められない場合は自動インストールされているはずです。
- You have selected the macOS system install of Python,,,
  - という注意文が出る場合には「Do not show again」を選択。

PythonインタプリタのPATHを確認

- ターミナル.appを起動。
  - source ~/.venv/prog1/bin/activate
  - which python
  - 上記whichコマンドで出力される結果（=Pythonインタプリタの PATH）を範囲選択してコピー。

インタプリタの設定



- 動作確認用のスクリプトを作成。
  - Fileメニューから New File。
  - print(1) とか最低限Python動作確認できるぐらいのコードを記入。
  - Fileメニューから Save as... を選択。
  - ホームディレクトリ上の prog1 フォルダ内に、week3.py という名前で保存。
- インタプリタの設定。
  - ウィンドウ左下に Python インタプリタ（背景紫色部分）が表示されているので、そこをクリック。
  - 「Enter interpreter path...」をクリック。
  - 前のスライドでコピーしたPATHをペーストして確定。

通常実行（Run）

- 通常実行
  - Runメニューから Run without Debugging... を選択。
  - もしくは、右上の▷アイコンをクリック。
- 右下パネルにターミナルが現れ、そこに実行結果が出力されているはずです。
  - print(1) と書いていたなら「1」と出力されていればOK！

ここからおまけ（だけど便利）

- **デバッグ（Debug）**とは
  - コードの誤り・欠陥等のバグ（Bug）を修正すること。
- 代表的なデバッグ方法
  - print出力
    - 気になるところでその都度print出力する。
    - 中身は確認できるが、その都度記述する必要があるため手間ががかり、また増えすぎるとコード全体の見通しが悪くなる。
  - =>**デバッガ（debugger）**を使おう！
    - スクリプトをまとめて実行するのではなく、「5行目で一時停止して」、「1行だけ実行を進めてまた一時停止して」といった「一時停止」を伴いながら変数を確認することができる。
    - ターミナルからCUIなデバッガを使うこともできるが、不便なのでGUIなデバッガを提供している実行環境（今回はVS Code）を使おう。

デバッグ実行1：スクリプト例の準備

- 新規ファイルを作成しよう。
- コードを一通り書き終えてから保存するのではなく、**まっさらな状態で**まずファイル名を付けて保存しよう。
  - .py拡張子で保存することで、Pythonエディタとして機能するようになる。
- **手打ち入力**。編集中、自動で何かが補完されたり、自動で何かが出力されたりするはずだ。
  - 一度宣言した変数・関数名等は、その後自動補完対象となる。
  - 標準関数（print, range）等は、記入時にその簡易ヘルプが出力され、使い方を確認できる。

```
# y = 2x + 1
def func(x):
    y = 2 * x + 1
    return y

for x in range(-2, 3):
    y  = func(x)
    print(x, y)
```

入力終えたら通常実行して動作確認しよう。通常は想定通りに動かないときにデバッグする。今回は、「デバッグ実行」に慣れることを優先するため、事前に動作することを確認しよう。

デバッグ実行2：ブレイクポイントの設定

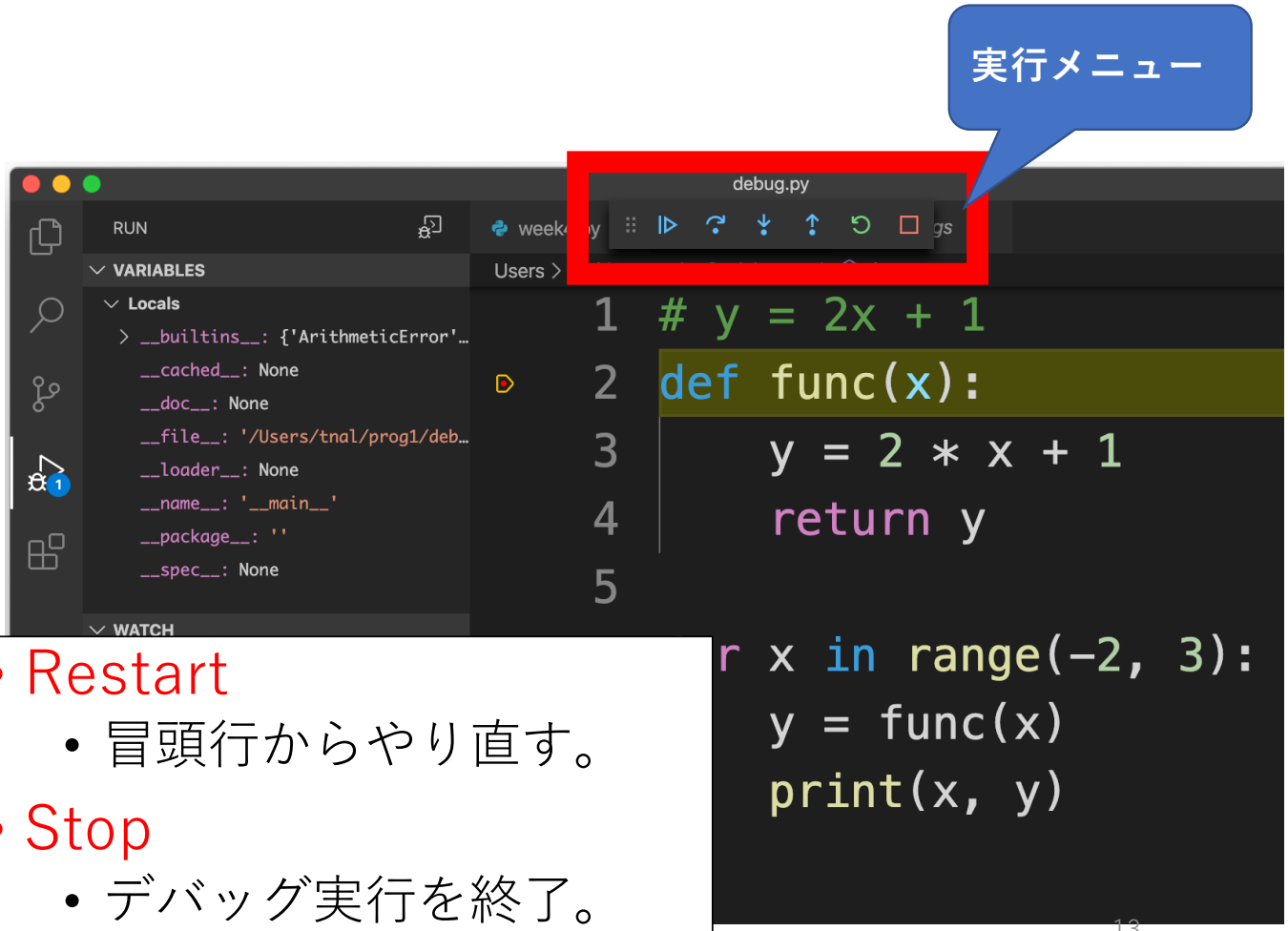
- ブレイクポイント（**breakpoints**）
  - 実行を一時停止させる場所のこと。複数設定することも可能。
- 設定方法
  - コード中の「行番号の左にある空きスペース」をクリック。**赤丸**が付いたらブレイクポイントを設定したことになる。
  - 赤丸をクリックすると、取り消すことができる。
  - 1行目のコメント行や、5行目の空白行のように何も実行する命令文がない行には設定できない。

デバッグ実行3：デバッグ実行する1

- Runメニューから **Start Debugging**を選択。
- デバッグ設定の選択肢が表示されるので、**Python File**を選択。
- 右画面のようになればOK。
  - 背景薄黄色の行が、「これから実行する行」。この行で一時停止している。
  - 左パネルのVariablesは変数で、スクリプト実行直後の時点で自動的に設定されている特殊な変数が既にある。

デバッグ実行4：デバッグ実行メニューの説明

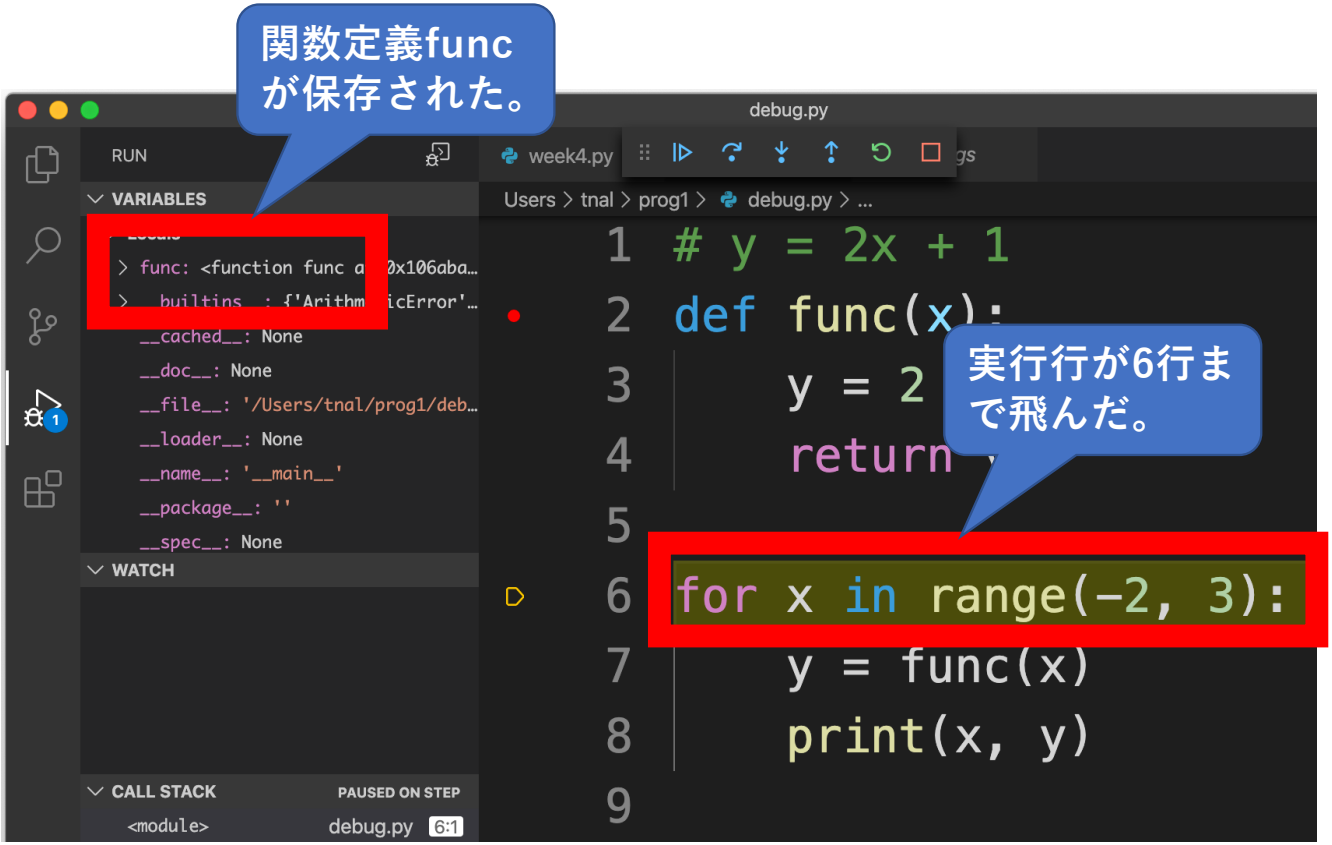
- 一時停止してる行から実行する方法が複数用意されている。
- Continue
  - 次のbreakpointまで実行継続。
- Step over
  - 今の行を実行して、次の行で停止。
- Step into
  - 今の行を実行して、次の行で停止。ただし関数呼び出しだった場合、関数の中に移動して停止。
- Step out
  - 今いる行が関数の中にある場合、その関数を呼び出している場所まで実行し続けてから停止。



- Restart
  - 冒頭行からやり直す。
- Stop
  - デバッグ実行を終了。

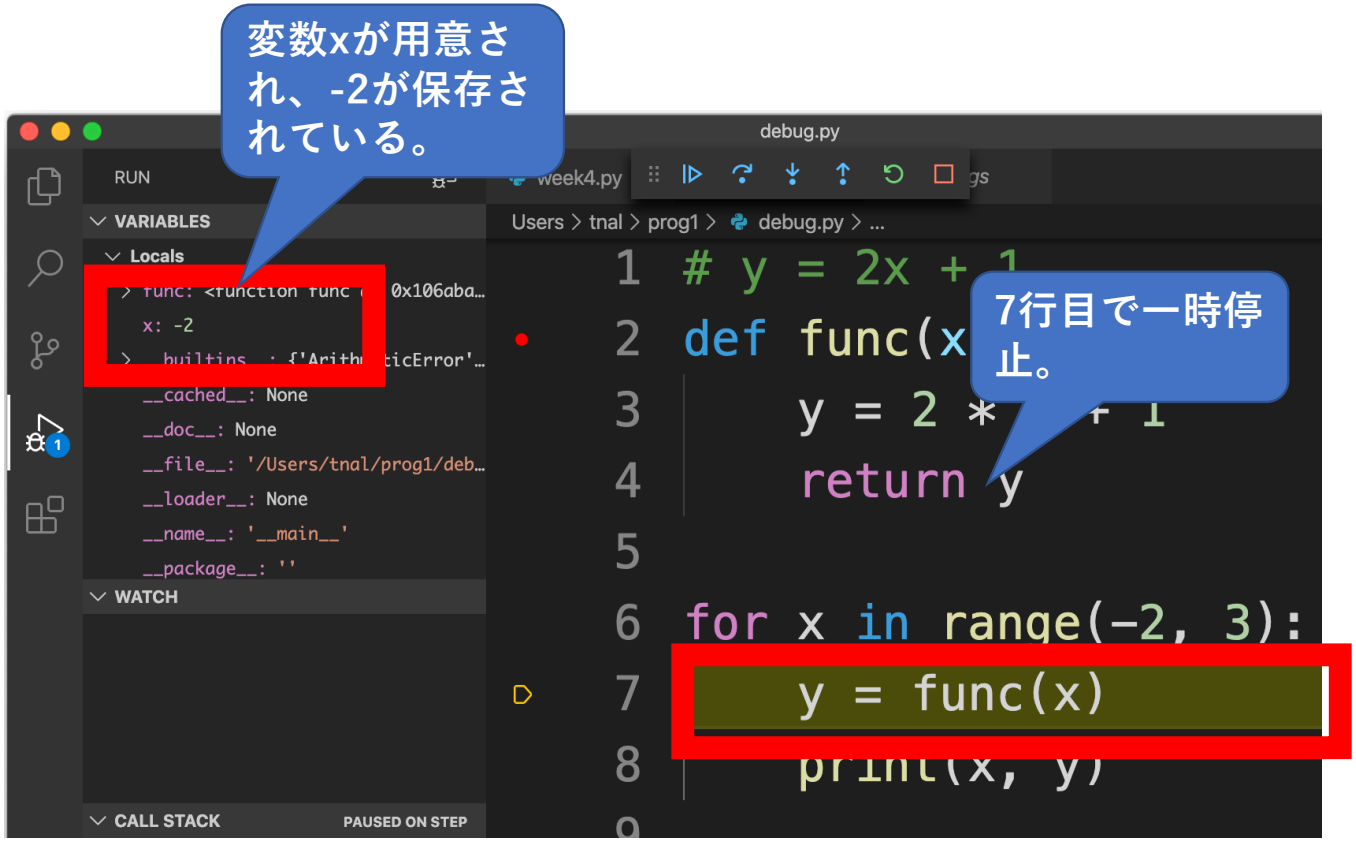
デバッグ実行5：Step into1回目

- 今回はStep intoだけに慣れよう。
- 一度クリックすると、3行目ではなく6行目に移動して停止する。これは2～4行目が関数定義であり、実行する命令文ではないため。また関数宣言を読み込んだ後は、メモリ上にfuncという関数が保存されており、この後は呼び出し実行することができるようになる。これが「関数定義」の意味。



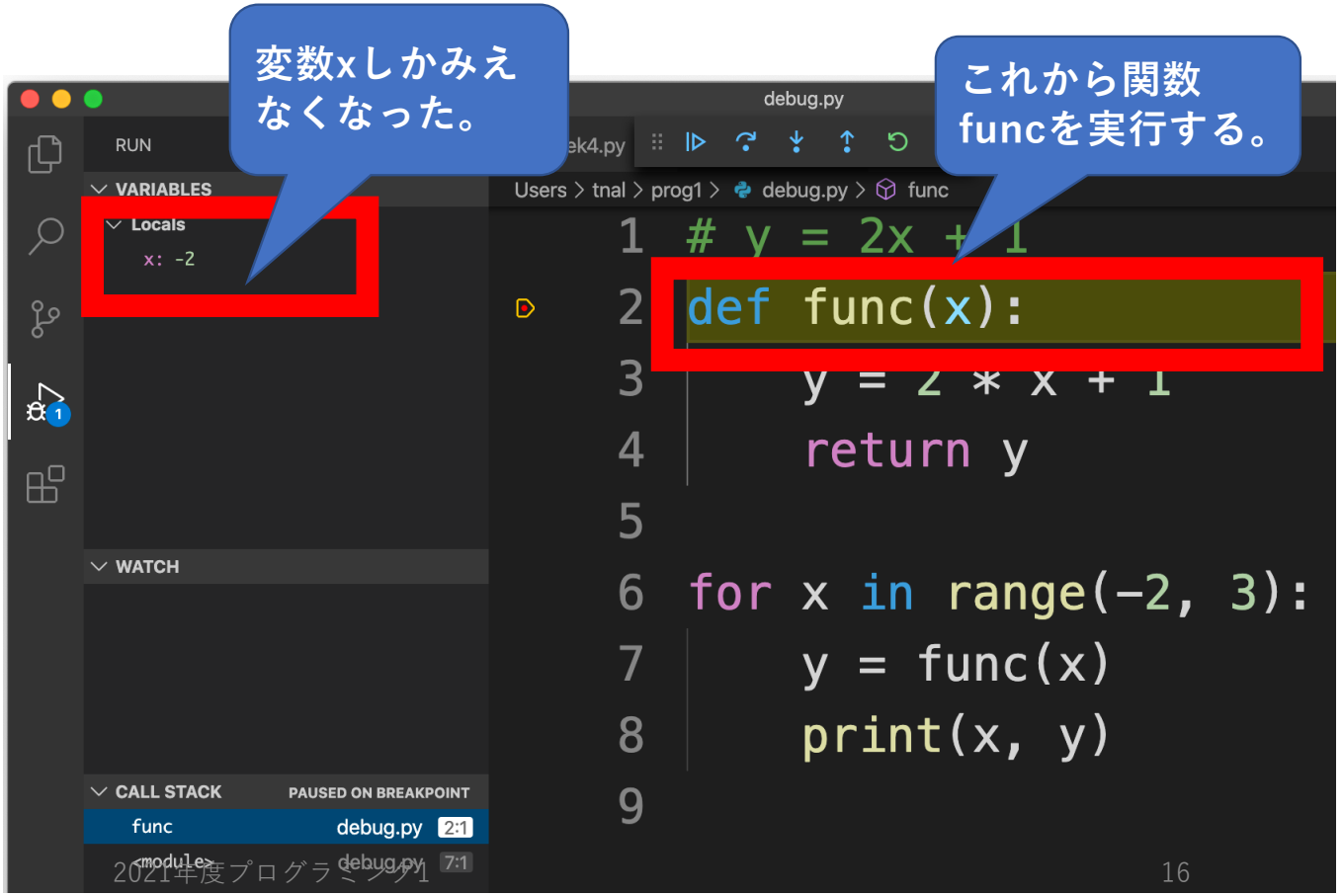
デバッグ実行6：Step into2回目

- Step into 2回目。
- 6行目を実行すると、range(-2, 3)により-2,-1,0,1,2という5個の要素で構成されるシーケンスが用意される。その先頭の要素である-2を変数xに保存し、ループブロックを実行しようとする（この後）。
- ということを**printせずとも確認できる**！



デバッグ実行7：Step into3回目

- Step into 3回目。
- 7行目を実行しようとすると、関数funcの呼び出し命令が書かれている。このため、関数funcの定義に移動し、そこで一時停止する。
- なお、その前のvariablesにはいくつかの変数や関数が記録されていたが、関数func内に入った時点で見えなくなる筈だ。このような変数が見える範囲を「**スコープ**」と呼ぶ。詳細は後日。



デバッグ実行終了、まとめ

- step into実行とは、
  - 一行ずつ処理して一時停止する。関数呼び出しならその中に移動して一時停止する。Return文や関数ブロック末端まで来ると、関数呼び出し元に戻って一時停止する。
  - いちいちprint書く必要がないので便利だし、コードが汚れることも少なくなる。
- デバッグ実行中は、
  - 一時停止しているだけで、メモリ消費し続けている（から現時点での変数の中身を確認できたりする）。デバッグ実行そのものを終了させるにはStopしよう。VS Code自体を終了してもOK。
- Step over, step outも試してみよう。
- Continue を試すには、ループ内か関数内どちらかにbreakpointを設定しないと、他との違いがよく分かりません。（何故か考えてみよう）