

Python 開発環境の準備と使い方

Python のプログラム開発環境には様々なものがあるが、ここでは Python に付属している IDLE (Integrated DeveLopment Environment), プログラミング支援機能を追加して効率的に作業ができる Visual Studio Code (VSCoDe), プログラムとともに説明文, グラフ, 実行結果などをまとめて保存できる JupyterLab, これらをまとめてインストールできる Anaconda, インストール不要の paiza.IO, 及び Google Colaboratory について紹介する。

IDLE や VS Code, paiza.IO は拡張子が py の Python のプログラムファイルを作成することができる。これに対して JupyterLab と Google Colaboratory は拡張子が ipynb のノートブック形式のファイルを作成することができる。

ただし, JupyterLab や VSCoDe はエディタであるため, Python もインストールする必要がある。IDLE だけでも十分にプログラムの開発ができるが, VS Code や JupyterLab などを追加することで, より効率的で多様な開発環境を整えることができる。

なお Anaconda は Python とともに JupyterLab や VSCoDe, 外部モジュールなどもまとめてインストールでき環境構築が容易であるが, 不要なものまで導入されて動作が重くなることもある。このような場合には個別にインストールするほうがよい。

インストールしないオンライン環境は便利であるが, 一部のプログラムが動かない場合やネットワーク接続に問題が発生すると利用できない場合があるので, できるだけインストールするオフライン環境を整備しておくことをお勧めする。

1. IDLE

1-1. Python のインストール

以下の URL にアクセスして Python3 のインストーラをダウンロードする。

<https://www.python.org/>

なお, 最新バージョンは外部モジュールが対応していない場合があるので, 対応しているバージョンのを選んでインストールするとよい。

インストール時に表示される「Add Python xx to PATH」の横にチェックを入れる(☑)とよい。

Python インストール後に, コマンドプロンプトで「pip install 外部モジュール名」を実行して授業で使用する外部モジュールをインストールしておくといよい。なお, 情報 I・情報 II でインストールしておくといよいと思われる外部モジュールは以下の通りである。

matplotlib, numpy, pandas, scipy, Pillow,
scikit-learn, requests, japanize_matplotlib,
japanmap

1-2. IDLE (統合開発環境) の簡単な使い方

ここでは Python に付属している IDLE の使い方を紹介する。IDLE には, エディタ(編集), プログラムの実行, デバッグなどの機能を有する。PC のスタートメニューから Python フォルダ内の IDLE を選択すると, 図1のシェルウィンドウが開く。

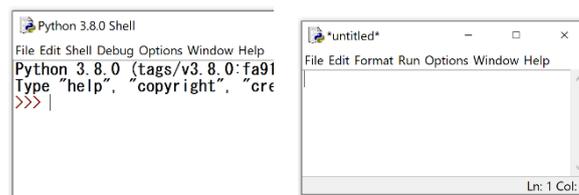


図 1 シェルウィンドウ 図 2 エディタウィンドウ

(1) 新規ファイルの作成

「File」, 「New File」の順に選択すると, 図2のエディタウィンドウが開く。エディタウィンドウに入力したプログラムをファイル名を付けて保存するには, 「File」, 「Save As」の順に選択する。

(2) 既存ファイルのオープン

「File」, 「Open」の順に選択する。

(3) プログラムの入力

プログラムはエディタウィンドウに入力する。

コロン「:」のあとで改行すると、以降は自動的に半角4文字分のインデントが設定される。「BackSpace」キーを押すとインデントが解除される。

(4) ファイルの上書き保存

「File」, 「Save」の順に選択する。

(5) プログラムの実行

「Run」, 「Run Module」の順に選択すると、実行結果はシェルウィンドウに表示される。

2. VSCode

2-1. VSCode のインストール

前記の Python に付属している IDLE だけでも十分にプログラムの開発ができるが、VS Code を使用すると、より効率的な開発が可能になる。

Python をインストールしたのち、以下の URL からダウンロードしてインストールする。

<https://code.visualstudio.com/>

インストール後のデフォルトのインデントと文字エンコードは、それぞれ「半角スペース 4 つ」と「UTF-8」になっているが、VSCode の画面下部でこれらを変更することができる。

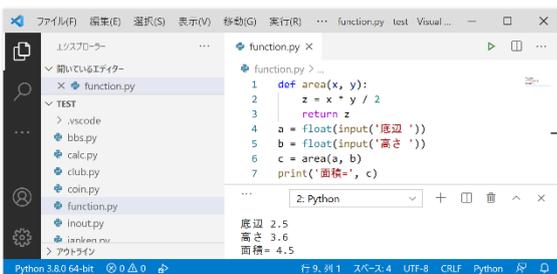


図 3 Visual Studio Code

(1) 日本語表示の拡張機能のインストール

上部のメニューから「View」, 「Extensions」の順に選択して表示される検索ボックスに「japanese」を入力すると、Microsoft の「Japanese Language Pack for Visual Studio Code」が表示されるので、その中の右下にある「Install」をクリックする。「Restart」をクリックして再起動すると、メニューなどが日本語表示される。

(2) Python の拡張機能のインストール

上部のメニューから「表示」, 「拡張機能」を選択して表示される検索ボックスに「Python」と入力すると、Microsoft の「Python」が表示されるので、その中の右下にある「インストール」をクリックすると、構文チェックなどの拡張機能が使えるようになる。

2-2. VSCode の簡単な使い方

(1) フォルダを開く

ファイルを保存するフォルダを選択するために、上部のメニューから「表示」, 「エクスプローラー」, 「ファイル」, 「フォルダを開く…」の順に選択すると、「フォルダを開く」というウィンドウが開く。

ファイルを保存するためのフォルダを選択してから「フォルダの選択」をクリックすると、左に選択したフォルダが表示される。新規フォルダを作成するには「新しいフォルダ」をクリックする。

(2) 新規ファイルの作成

上記 (1) の操作後に、「ファイル」, 「新規ファイル」の順に選択すると、上部に「Untitled-1」という仮のファイル名でタブが表示される。

続けて「ファイル」, 「名前を付けて保存…」の順に選択し、「ファイル名 .py」を入力し、「保存」をクリックする。以下の確認を行う。

① 左下に「Python 3.* *bit」が表示される場合

Python の実行が可能な状態になっている。

② 左下に「Select Python Interpreter」が表示される場合

「Select Python Interpreter」をクリックすると、上部にインストールされている Python のバージョン「Python 3.* *bit」が表示される。

上部の「Python 3.* *bit」をクリックすると、左下の「Select Python Interpreter」が「Python 3.* *bit」に切り替わり、Python の実行が可能な状態になる。

(3) 既存ファイルのオープン

上記 (1) の操作後に、上部のメニューから「ファイル」, 「ファイルを開く…」の順に選択し、ファイル(ファイル名.py)を選択してから、「開く」をクリックする。上記(2)の①・②の確認を行う。

(4) プログラムの入力

右上のエディタエリアにプログラムを入力する。

(5) ファイルの上書き保存

「ファイル」、「保存」の順に選択する。

(6) プログラムの実行

「実行」、「デバッグなしで実行」の順に選択すると、右下の「ターミナル」に実行結果が表示される。

3. JupyterLab

3-1. JupyterLab のインストール

前記の Python に付属している IDLE だけでも十分にプログラムの開発ができるが、JupyterLab を使用すると、Notebook 形式のファイルも扱うことができる。Python をインストールしたあと、コマンドプロンプトから以下のコマンドでインストールする。

```
pip install jupyterlab
```

3-2. JupyterLab の簡単な使い方

(1) 起動

コマンドプロンプトから以下のコマンド(jupyter と lab の間に半角スペース)を入力する。

```
jupyter lab
```

なお、Internet Explorer では正常動作しない。

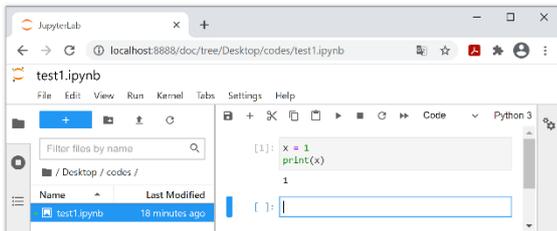


図 4 JupyterLab

(2) 作業フォルダの選択

左の領域(左サイドバー)が表示されていない場合は、左上の「」をクリックすると、左の領域が表示され、その中にフォルダやファイルの一覧が表示されるので、ファイルの読み書きをするためのフォルダ(作業フォルダ)を開いておく。

例えば、デスクトップを作業フォルダにするには、「Desktop」をダブルクリックする。新規にフォルダを作成する場合は「」をクリックする。

(3) ノートブックの新規作成

右の領域(ワークエリア)の「Notebook」の下の

「Python3」をクリックすると、コードを入力するための青い枠(セル)が表示される。

上の領域(メニューバー)から「File」、「Rename Notebook...」の順に選択し、ファイル名を入力し(.ipynb 以外の部分を修正する)、「Rename」をクリックすると、拡張子が ipynb のノートブック形式のファイルが左の領域に表示される。

(4) プログラムの入力例と実行例

右の領域の上部の「」の左に「code」が表示されていることを確認する。「code」以外の文字が表示されている場合は、「」の部分をクリックして「code」を選択する。

「code」になっている状態で、コードをセルの中に入力する。例えばセル内に「x = 1」を入力し、「Enter」を押すと改行される。

続けて「print(x)」と入力し、「Shift」キーを押しながら「Enter」キーを押す(以下、「Shift+Enter」と記す)か、もしくは上部にある「▶」ボタンをクリックすると、そのセルで実行した結果である「1」がセルの下に表示され、次のセルが追加される。セルの左端には[n] と表示され、n は実行時に連番が表示される。

(5) ノートブックの上書き保存

上の領域から「File」、「Save Notebook」の順に選択するとファイルが上書き保存される。

(6) 終了

「File」、「Shut Down」の順に選択し、「Shut Down」をクリックすると、「Server Stopped」と表示されるので、その後 Web ブラウザを閉じる。

(7) その他の操作

① ノートブックの読み込み

ノートブック(ipynb 形式)が格納されているフォルダを開き、そのファイルをダブルクリックすると、右の領域にノートブックが表示される。

② py 形式のファイルのセルへの読み込み

左の領域で、読み込むプログラムファイル(拡張子が py)が格納されているフォルダを開いておく。セル内に、「%load ファイル名.py」を入力して、「Shift+Enter」を押すと、セル内に「ファイル名.py」のファイルの内容が読み込まれる。なお、py 形式のファイルをメモ帳などのテキストエディタなどで開き、範囲を指定してコピーしたあと、JupyterLab の

セルの上で「Ctrl+V」で貼り付けをしてもセル内にプログラムを読み込むことができる。

③ py 形式のファイルの作成・保存

「File」, 「Export Notebook As...」, 「Executable Script」の順に選択し、保存するフォルダを選択して保存すると、セルに入っているプログラムが py 形式のファイルとして保存される。

④ エディットモードとコマンドモードの切替

セルをクリックすると、周りの枠線が青色になり、そのセルはプログラムや注釈などを入力するための「エディットモード」になる。また、セルの枠線の左側の部分をクリックすると、その中と周りの枠線が灰色になり、セルの移動などのセルそのものに対する操作を行うための「コマンドモード」になる。

⑤ セルの移動

セルの枠線の左の部分ドラッグする。

⑥ セルの追加

右の領域の上部の「+」をクリックする。

⑦ セルの削除

削除したいセルをクリックし、右の領域の上部のはさみの形をしたボタン「✂」をクリックする。

⑧ すべてのセルの実行

「Run」, 「Run All Cells」の順に選択する。

⑨ 注釈の入力

注釈を入力するセルをクリックして、右の領域の上部の「Code」の右にある「▼」をクリックして「Markdown」に変更する。

「Markdown」になっている状態でセルの中に注釈の文字列を入力し、「Shift+Enter」を押す。なお、注釈文字は、そのまま入力してもよいが、「# 文字列」や「** 文字列 **」などの Markdown 記法で入力すると、文字の大きさや太字などの装飾ができる。

⑩ カーネルの再起動

Jupyter Lab では、以前に実行したセルで定義した変数や関数は、別のセルでも使用することができるが、実行結果が別のセルに影響を与えると不都合なこともある。

また、セルの実行の順番を変えたり、削除したりすると変数の値などが想定外の状態となり、実行結果が思い通りにならないことがある。そのような場合は、上の領域から「Kernel」, 「Restart Kernel」,

「Restart」の順に選択してカーネル(プログラムを実行する機能)の再起動を行うと、セルを実行していない最初の状態に戻ることができる。

4. Anaconda

以下 からダウンロードしてインストールする。

<https://www.anaconda.com/>

インストール時に「Add Anaconda 3 to the system PATH environment variable」にを入れる。

1. で述べた外部モジュールのうち、`jupyterlab` と `jupyterlab_matplotlib` 以外は Anaconda のインストールとともに自動的にインストールされる。

(1) JupyterLab の起動

スタートメニューの「Anaconda3」フォルダ内の「Anaconda-Navigator(Anaconda3)」を選択し、「JupyterLab」の「Launch」をクリックする。

(2) IDLE の起動

スタートメニューの「Anaconda3」フォルダ内の「Anaconda Powershell Prompt(anaconda3)」または「Anaconda Prompt(anaconda3)」を起動し、「idle」と入力する。

5. Google Colaboratory

Google アカウントでログインして利用する。

<https://colab.research.google.com/>

Google Colaboratory は、JupyterLab とは細かな違いはあるが、ほぼ同じ操作性を有している。主な特徴は以下の通りである。

(1) メリット

インストールなどの環境構築はほぼ必要がなく、無料で利用でき、共有が簡単である。

(2) デメリット

90 分経過すると実行環境が初期化され、12 時間経過するとノートブック以外のファイルが削除される(Google ドライブと連携することで回避できる)。

PC のファイルはアップロードする必要がある。

また、Tkinter (GUI を構築・操作するための標準モジュール)などを使ったローカル環境で動作するプログラムは動作しない。

6. paiza.IO

paiza.IO (パイザアイオー)はオンラインでプログラミングができる実行環境である。

無料でユーザ登録は不要であるが、登録しても無料で使え、自分が作成したコードの保存や読み出し、削除などができる。

なお、グラフ描画のための `matplotlib` モジュールや GUI アプリの作成のための `Tkinter` モジュールなどには対応していないため、これらのモジュールを使用したプログラムは動作しない。

(1) プログラムの作成と実行

以下の URL に接続する。

<https://paiza.io/ja>

「コード生成を試してみる(無料)」をクリックする。

左上の緑のボタンにマウスポインタを重ねると、言語の一覧が表示されるので、「python3」をクリックすると、図5のようになる。



図5 paiza.IO

「Main.py」と表示されている部分をダブルクリックすると、文字列を変更できる状態になる。

「Main.py」の「Main」の部分をも、例えば「test1」に変更し「Enter」を押して、「test1.py」にする。

次に中央のプログラム入力欄にプログラムを入力し、「実行(Ctrl-Enter)」をクリックすると、画面下部に実行結果が表示される。

なお、キー入力を伴うプログラムの場合は、実行する前に画面下部の「入力」タブをクリックし、その下の入力欄にキー入力するデータを入力してから「実行(Ctrl-Enter)」をクリックする。入力するデータが複数の場合は、データを改行して複数入力する。

プログラムにバグがある場合は、画面下部に「実

行時エラー」タブが表示され、その下にエラーメッセージが表示される。

(2) ユーザ登録

右上の「サインインアップ」をクリックする。「ユーザー名」、「メールアドレス」、「パスワード」を入力し、「サインアップ」をクリックする。登録したメールアドレスにメールが届くので、「アカウントを確認する」をクリックする。「アカウントを登録しました。」と表示される。

(3) ログイン

右上の「ログイン」をクリックし、「メールアドレス」と「パスワード」を入力し、「ログイン」をクリックする。

(4) ログアウト

ログイン状態で、人の形のアイコン  の上にマウスポインタを重ね、「ログアウト」をクリックする。

(5) ファイルの保存

プログラムを入力して「実行(Ctrl-Enter)」をクリックすると、実行と同時に保存される。保存だけ行う場合は、「実行(Ctrl-Enter)」の横の「△」をクリックし「Save only」をクリックする。

(6) ファイルの読み出し

ログイン状態で「一覧」をクリックする。

「自分のコード」、「All」の順にクリックすると、保存したプログラムが表示される。「(python3)」をクリックすると、編集・実行できる状態になる。

(7) ファイルのアップロード

「新規コード」をクリックし、「Main.py」の横の「×」をクリックして「Main.py」を削除し、拡張子が `py` のファイルをプログラム入力欄にドラッグ&ドロップする。なお、ファイル名にスペースなどの使用不可の文字が含まれていると動作しない場合がある。

(8) ファイルの削除

ログイン状態で「一覧」をクリックする。「自分のコード」、「All」の順にクリックすると、保存したプログラムが表示される。「(python3)」をクリックすると、編集・実行できる状態になる。

「実行(Ctrl-Enter)」の横の「△」をクリックし「Delete」、「OK」の順にクリックする。

(9) 新規コードの作成

プログラムが表示されている状態で、これとは異なるプログラムを新規に作成する場合は、「新規コード」をクリックする。

7. IDLE によるデバッグ

デバッグ、プログラムの欠陥(バグ)を発見・修正するデバッグ作業を支援するソフトウェアである。

デバッガを使うと、ブレークポイント(設定した場所で処理を一時停止させる機能)やステップ実行(プログラムを1行ずつ実行する機能)などを使って、変数の値を調べたり関数の呼び出し履歴をみたりできるので、問題箇所を発見することができる。

① デバッガの起動

Python の IDLE を起動し、図1のシェルウィンドウのメニューから、「Debug」、「Debugger」の順にクリックする。図6のデバッガウィンドウが開くので、「Stack」、「Source」、「Locals」、「Globals」の横にある4つのチェックボックスにすべてチェックを入れる。

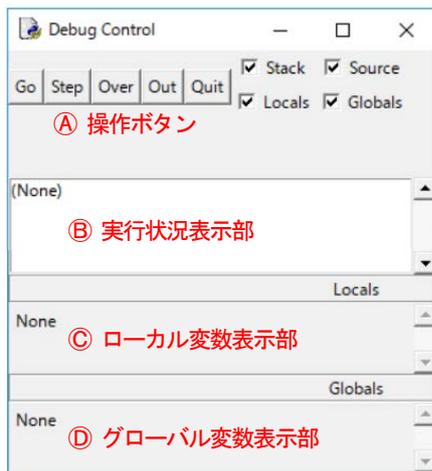


図6 デバッガウィンドウ

図6のAの5つのボタンを操作しながらデバッグを行う。

・ Go

プログラムをブレークポイントの直前まで実行する。ブレークポイントを設定しない場合は、最後まで実行される。

・ Step

1行ずつ実行する(現在の行を実行して、次の行で停止する)。現在の行に関数の呼び出しがあると、関数の中に入ったところで停止する。

・ Over

1行ずつ実行する(現在の行を実行して、次の行で停止する)。現在の行に関数呼び出しがあると、関数を実行してから戻ったところで停止する。

・ Out

関数内で停止している場合に、関数を最後まで実行してから戻ったところで停止する。

・ Quit

デバッグを終了する。

② デバッグの方法

デバッグの方法には、1行ずつ実行しながら確認するステップ実行とブレークポイントを設定して特定の行を確認する方法がある。

・ ステップ実行

ここでは、図7のプログラムの中の各変数の値の変化を、Python 用の統合開発環境である IDLE に内蔵されているデバッガ機能を使って調べる方法について説明する。

デバッガウィンドウを開いた状態で、エディタウィンドウのメニューから、「Run」、「Run Module」の順にクリックしてプログラムを実行すると、図8のように①行目の「def add(x,y)」が網掛けされる。

この状態からデバッガウィンドウの「Step」ボタンをクリックすると、図9のように④行目の「a = 3」が網掛けされる。

```
① def add(x, y):
②     z = x + y
③     return z
④ a = 3
⑤ b = 5
⑥ c = add(a, b)
⑦ print(c)
```

図7 プログラム

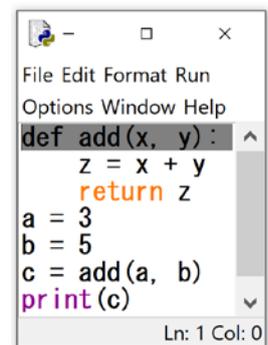


図8 図7の①行目

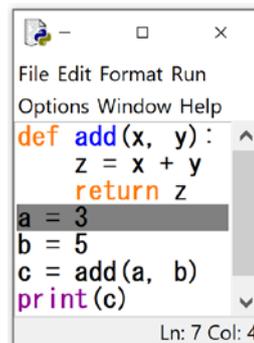


図9 図7の④行目

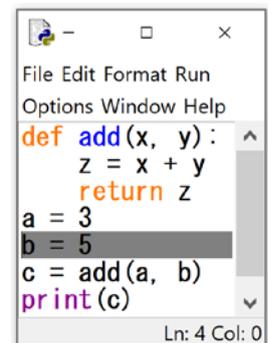


図10 図7の⑤行目

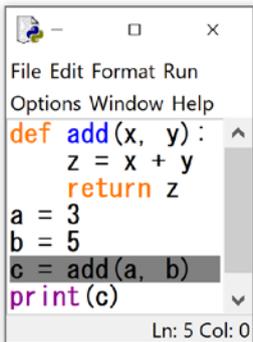


図 11 図 7 の⑥行目

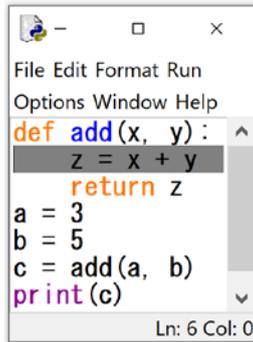


図 12 図 7 の②行目

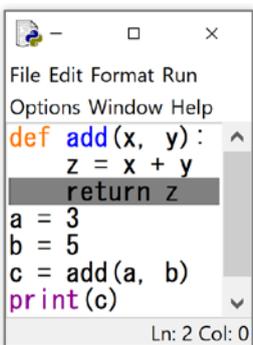


図 13 図 7 の③行目

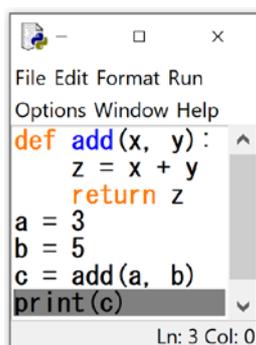


図 14 図 7 の⑦行目

「def add(x,y)」は関数定義の部分であり、⑥行目の関数の読み出しが実行されていないため、関数の内部に入らず、④行目に飛ぶことになる。

なお、網掛け部分のコードはまだ実行されていないことを示しているため、「a = 3」が網掛けになっている段階では変数 a に 3 は代入されていない。

さらに「Step」をクリックすると④行目のコードが実行され a に 3 が代入されて図 10 のように⑤行目の「b = 5」が網掛けされる。このように「step」ボタンをクリックしていくと、図8→図9→図 10→図 11→図 12→図 13→図 14 の順に網掛けが移動していく。

表 1 表示される変数の値

網掛けの表示順 (図 7 の番号)	ローカル変数 (図 6 の㉔)			グローバル変数 (図 6 の㉕)		
	x	y	z	a	b	c
①						
④						
⑤				3		
⑥				3	5	
②	3	5		3	5	
③	3	5	8	3	5	
⑦				3	5	8

表1の左端列の番号は「step」ボタンをクリックするごとに、網掛けの部分が図7の①→④→⑤→⑥→②→③→⑦の順に変化することを表している。

2列目と3列目は、図6の㉔と㉕の部分に表示されるローカル変数とグローバル変数の値をそれぞれ表している。⑦行目が網掛けになった状態で「Over」ボタンをクリックすると「print(c)」が実行され、シェルウィンドウに計算結果の 8 が表示される。

・ブレイクポイントの設定

ここではブレイクポイントを設定してデバッグする方法について説明する。

図 15 のように「c = add(a, b)」の部分で右クリックして「Set Breakpoint」をクリックすると、ブレイクポイントが設定され黄色い網掛け状態になる。

図5のデバッガウィンドウを開いた状態で、エディタウィンドウのメニューから、「Run」、「Run Module」の順にクリックしてプログラムを実行する。

「Go」をクリックすると、ブレイクポイントを設置した「c = add(a, b)」まで一挙に実行して停止しする。

その後、「Step」や「Over」ボタンをクリックしてステップ実行でデバッグを進めていく。ブレイクポイントの解除は、ブレイクポイントを設定した行で右クリックして、「Clear Breakpoint」をクリックする。

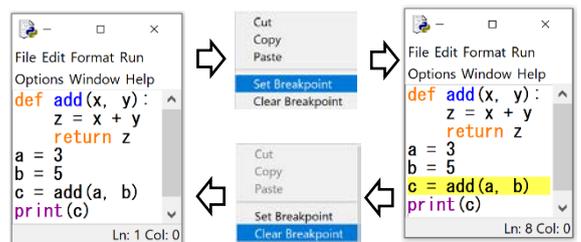


図 15 ブレイクポイントの設定と解除

③エラー表示

例えば、図7の⑥行目を「c = add(a)」に変更して実行すると、デバッガウィンドウには図16のエラーメッセージが表示される。また、シェルウィンドウにも同様のメッセージが表示される。

```
debug1.py:6: <module>()
TypeError: add() missing 1 required positional argument: 'y'
'bdb'.run(), line 587: exec(cmd, globals, locals)
> '._main_'.<module>(), line 6: c = add(a)
```

図 16 エラーメッセージ