

Pythonデバッグ手法

神戸デジタル・ラボ

長山哲也

長山 哲也

セキュリティコンサルタント

Python大好き

DJ・トラックメイカー

@Euphoricwavism





Description

Python用のIDEを利用してデ
バッグしてみよう

アジェンダ

- デバッグとは
- IDEを使ったデバッグ
- ハンズオン



デバッグとは

デバッグとバグの種類

デバッグとは、コンピュータのプログラムのバグを見つけ、手直しをすること。

アルゴリズムのバグ

条件式の誤りや利用する関数の間違い等により、想定したルートや処理ができていないバグ

今回の対象範囲

データのバグ

アルゴリズムは正しいものの、前提条件となる変数に入っているデータそのものに誤りのあるバグ

リソースのバグ

アルゴリズムやデータは正しいものの、計算量が多い等の理由でCPUやメモリ等のハードウェア資源を枯渇させるバグ

デバッグの手段

今回は・・・

- ・ブレークポイントとステップ実行による確認
- ・変数のウォッチにより想定通りの値になっているか確認

(用語) ブレークポイント

ブレークポイント

実行中のプログラムを意図的に一時停止させる箇所。問題となりそうな箇所の手前に設置し、そこからステップ実行させる等することが可能となる。

条件付きブレークポイント

変数等がある状態になった時に一時停止させるためのブレークポイント。

例外時ブレークポイント

処理が例外をスローした時点で一時停止させるためのブレークポイント。

(用語) ステップ実行

ステップ実行	1ステップ（1行ずつ）実行すること。
ステップイン	1ステップだけ実行すること。その際にメソッド等の実行があれば、そのメソッド内のステップを1ステップ実行する。
ステップオーバー	1ステップだけ実行するが、その際にメソッド等の実行がある場合、そのメソッドは一つの処理として全て実行した状態で1ステップとする。
ステップアウト	メソッド内でステップ実行している際に、その関数の終わりまで処理を一気に進め、呼出元に戻ることに。


(用語) 変数のウォッチ

変数のウォッチ

ブレークポイント等でプログラムの実行を一時中断した時点での変数の状態を確認すること。

ウォッチ式

変数を式に代入した時の状態を確認できる。変数がメソッドの引数となった時の戻り値や、条件式に代入された際の真偽を確認する際に有用。



IDEを使った デバッグ

代表的なデバッガ

デバッグツール	説明
Pdb	Python標準のデバッガ
PuDB	CUI形式でのPythonのデバッガ
PyCharm	JetBeansが出しているPythonのIDE環境
PyDEV	IBMが開発したIDE、EclipseのPythonプラグイン



PyDEVの場合

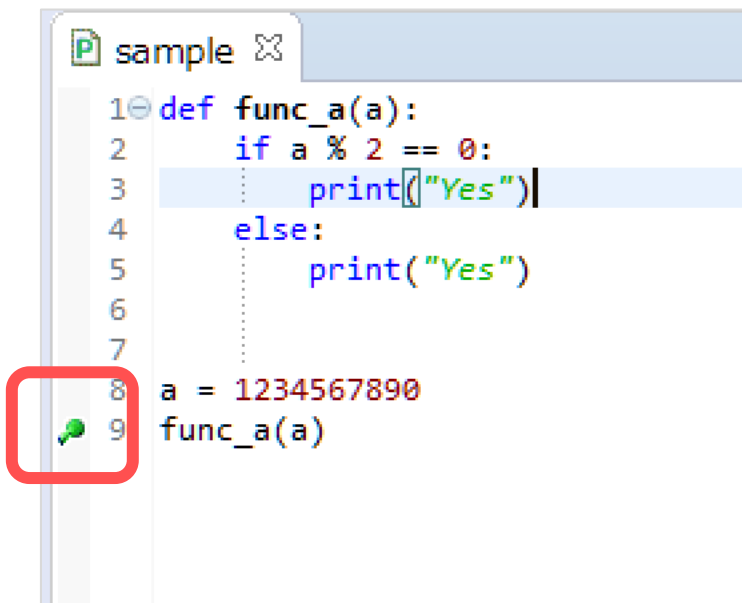
PyDevの場合

準備（各種インストール）

JDKのインストール	http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
Eclipseのインストール	https://www.eclipse.org/downloads/
Pleiadesのインストール	http://mergedoc.osdn.jp/
PyDdevのインストール	http://www.hiskip.com/pg-notes/lang/python/pythondevenv/394.html

PyDevの場合

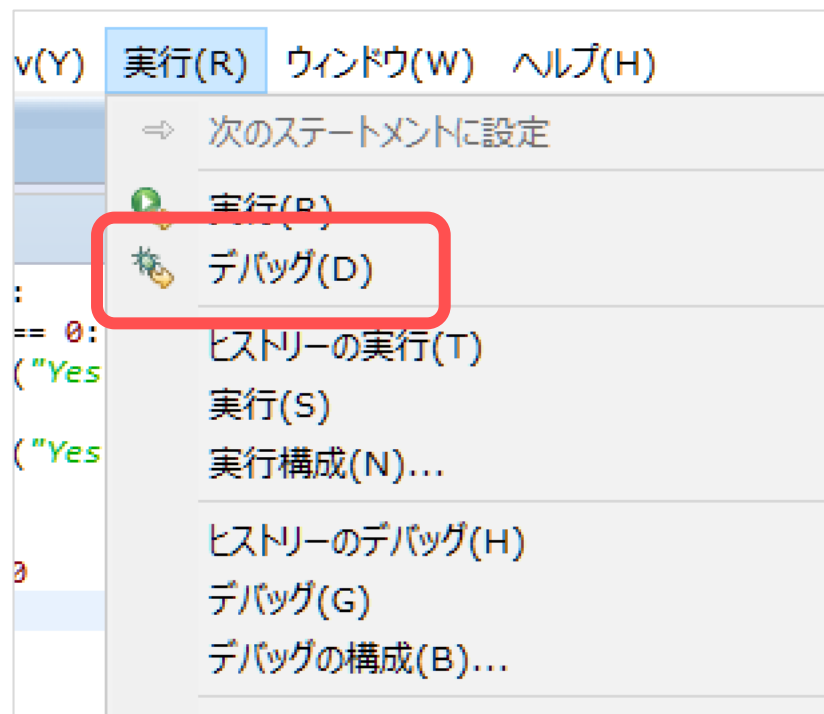
ブレークポイントの設置とデバッグの実行



The image shows a PyDev editor window with a file named 'sample'. The code is as follows:

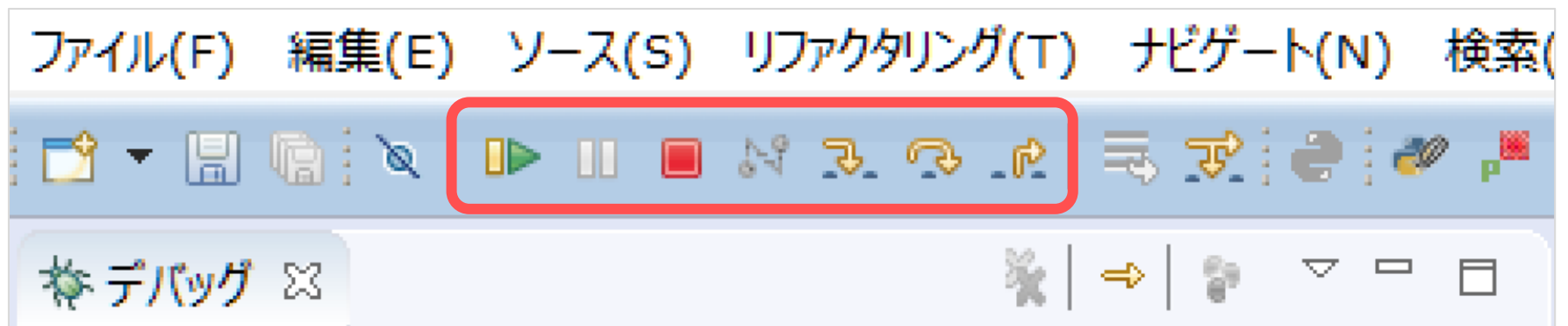
```
1 def func_a(a):  
2     if a % 2 == 0:  
3         print("Yes")  
4     else:  
5         print("Yes")  
6  
7  
8 a = 1234567890  
9 func_a(a)
```

A red box highlights a green bug icon in the left margin next to line 8, indicating a breakpoint is set at that line.



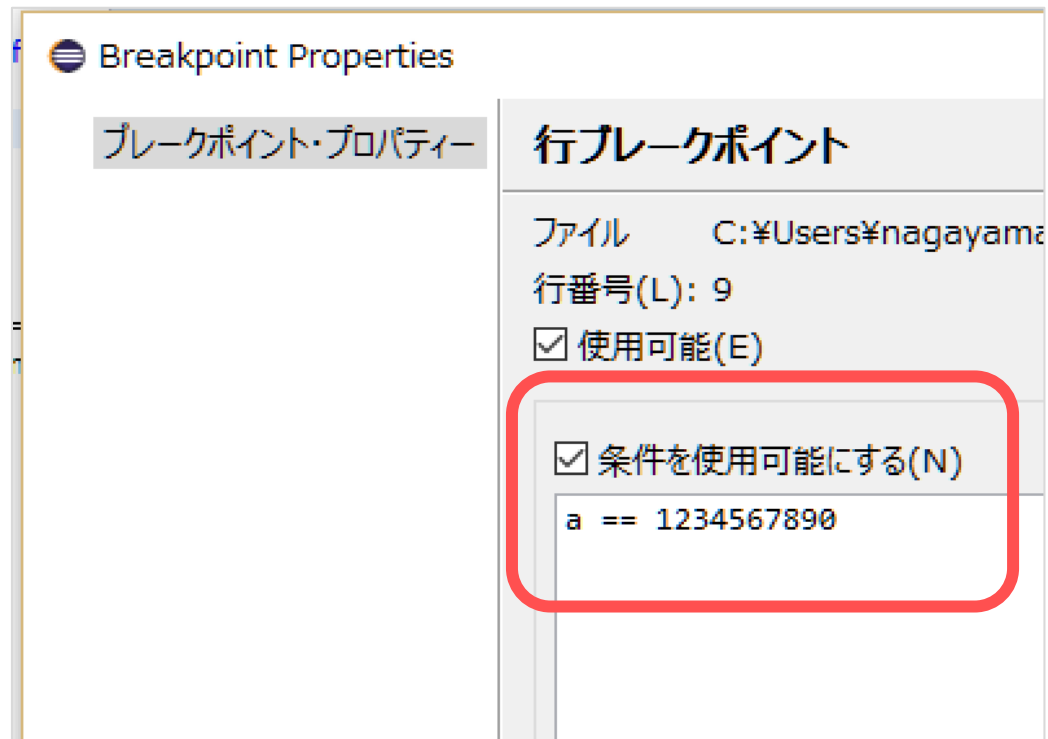
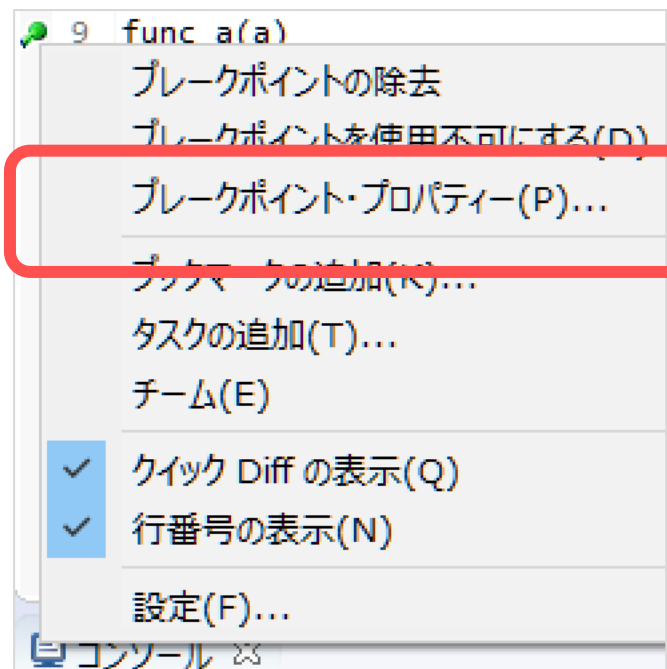
PyDevの場合

ステップ実行



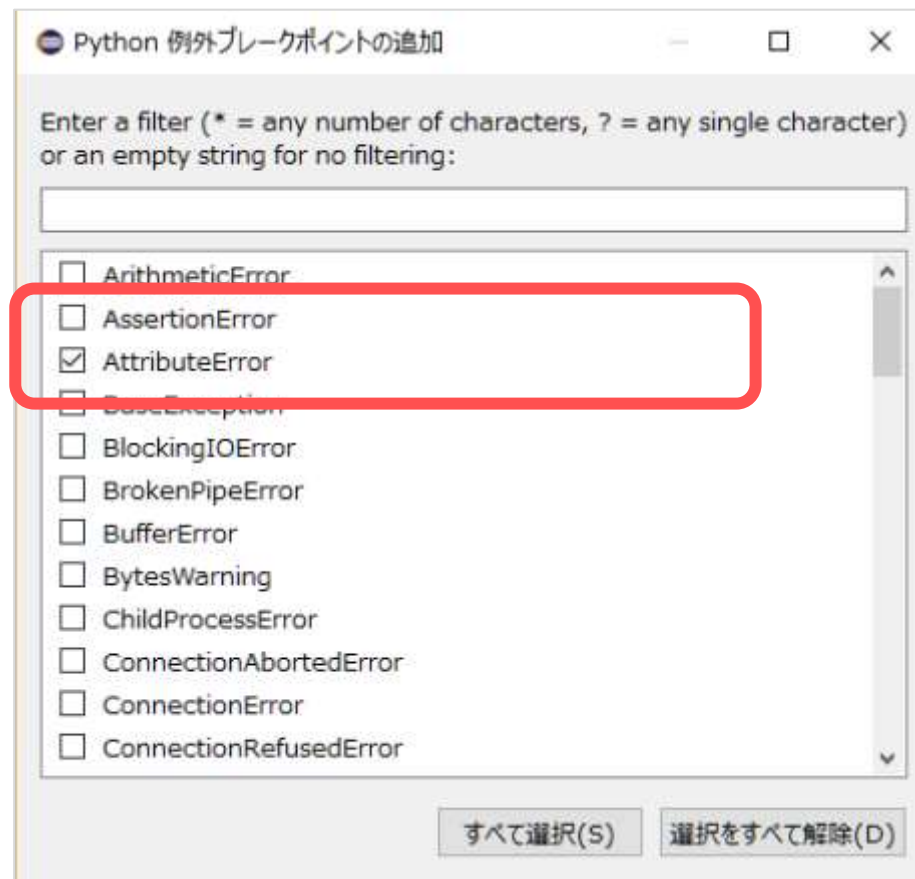
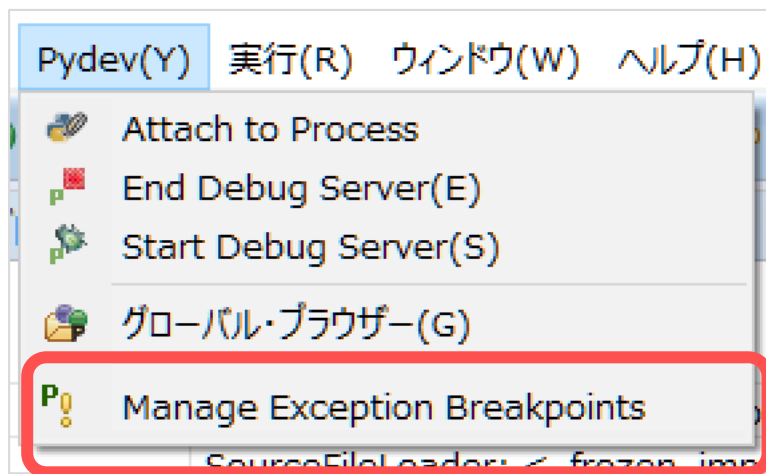
PyDevの場合

条件付きブレークポイントの設定



PyDevの場合

例外時のブレークポイントの設定



PyDevの場合

変数のウォッチとウォッチ式の追加

(x)= 変数 ブレークポイント 式	
名前	値
• __doc__	NoneType: None
• __file__	str: C:\¥¥Users¥¥nag
> • __loader__	SourceFileLoader: <
• __name__	str: __main__
• __package__	NoneType: None
• __spec__	NoneType: None
• a	int: 1234567890
• func_a	function: <function f

(x)= 変数 ブレークポイント 式	
名前	値
x+y =? "a - 1"	int: 1234567889
+ Add new expression	

A large, thin brown circle centered on the slide, containing the text "PyCharmの場合". The background features abstract orange and yellow geometric shapes at the top and bottom.

PyCharmの場合

PyCharmの場合

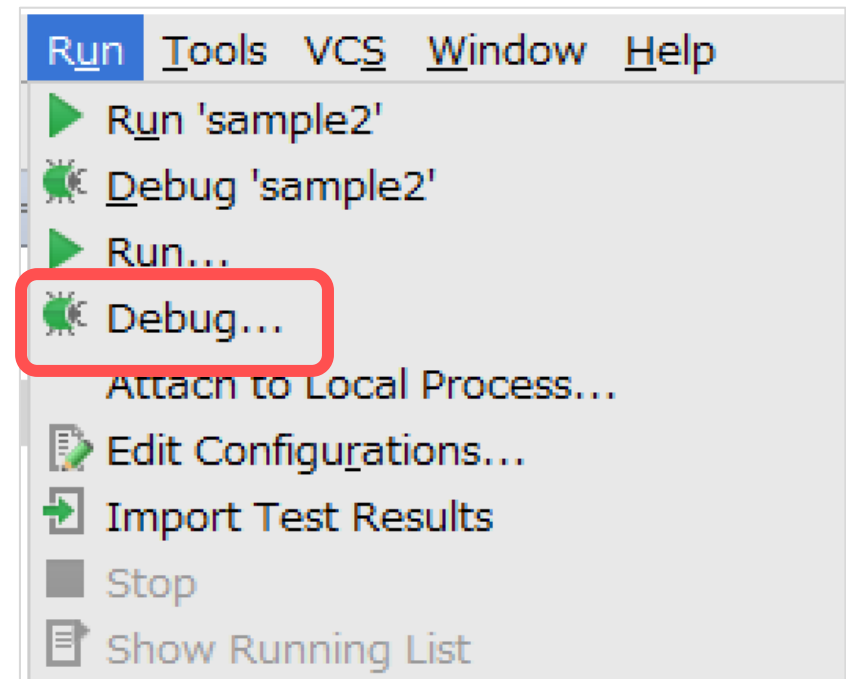
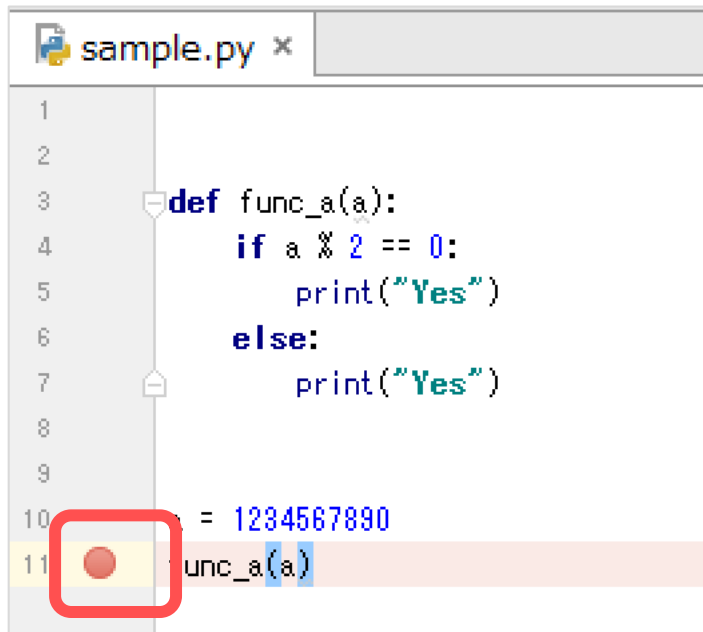
準備（各種インストール）

PyCharmのインストール

<https://www.jetbrains.com/pycharm/download/>

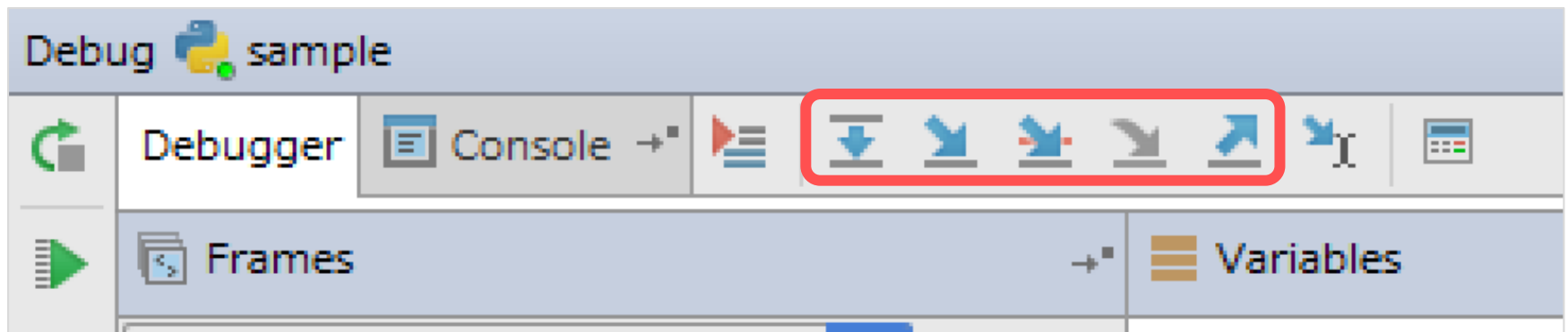
PyCharmの場合

ブレークポイントの設置とデバッグの実行



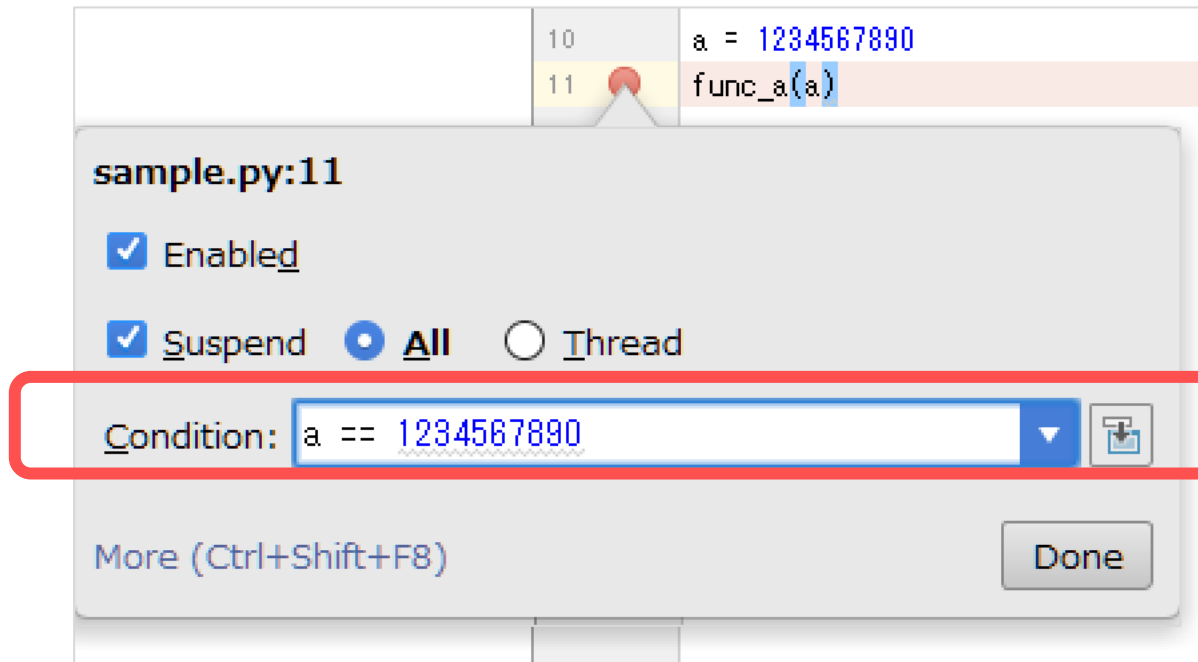
PyCharmの場合

ステップ実行



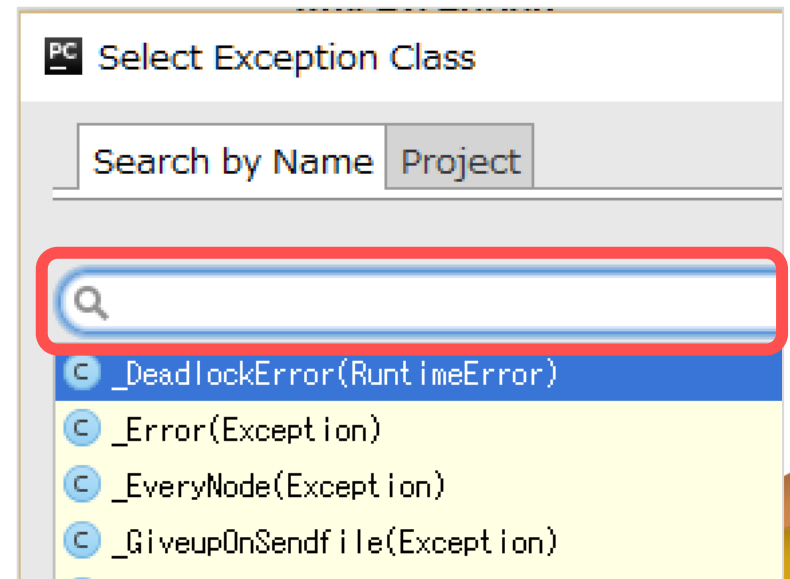
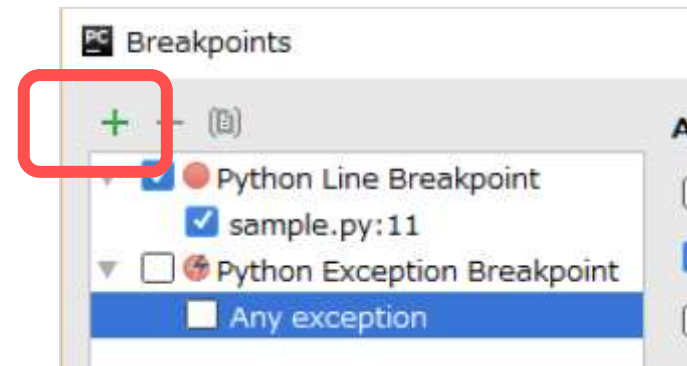
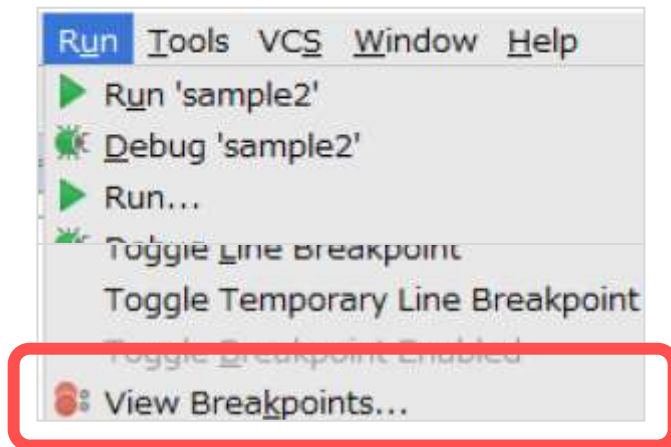
PyCharmの場合

条件付きブレークポイントの設定



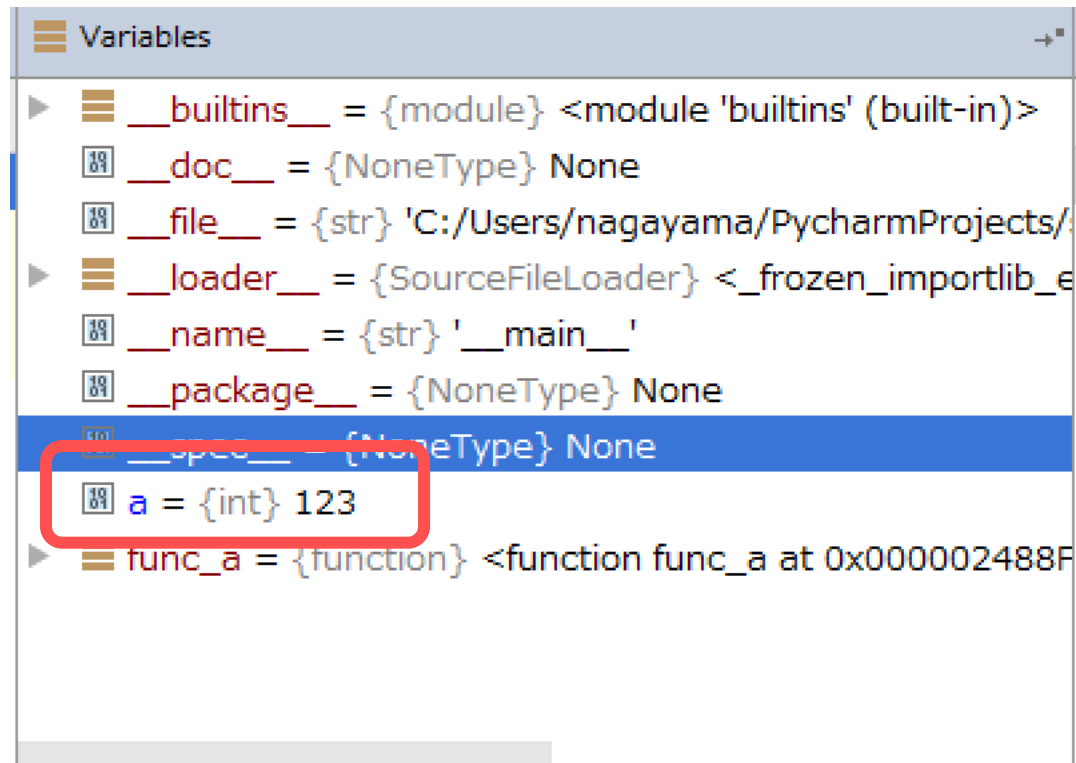
PyCharmの場合

例外時のブレークポイントの設定



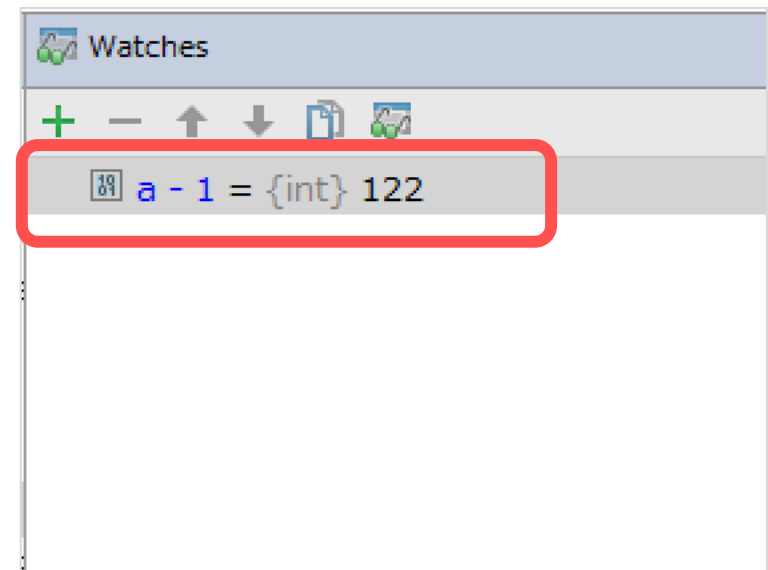
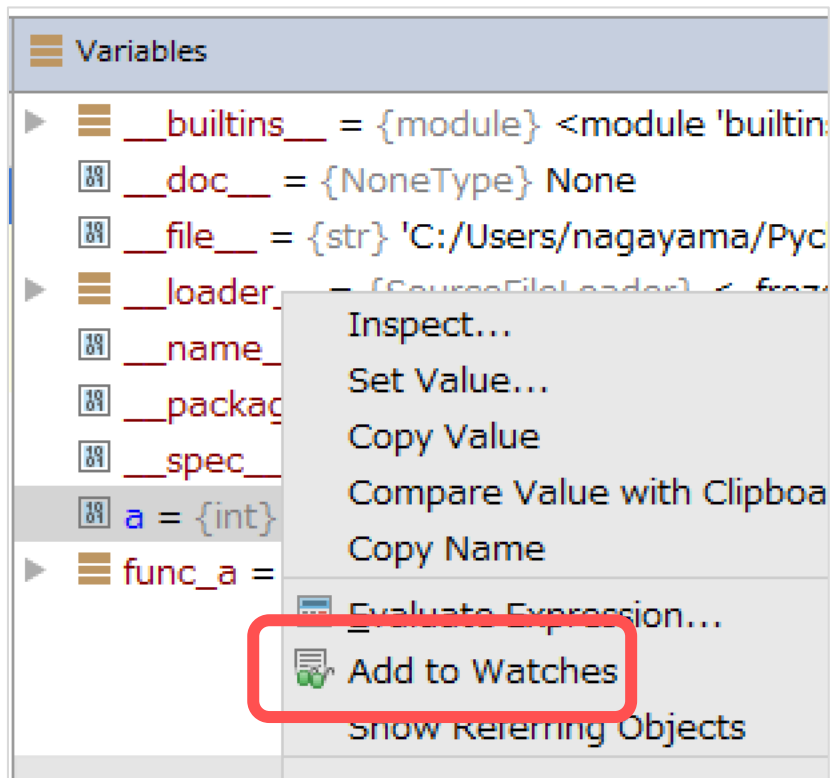
PyCharmの場合

変数のウォッチ



PyCharmの場合

ウォッチ式の追加

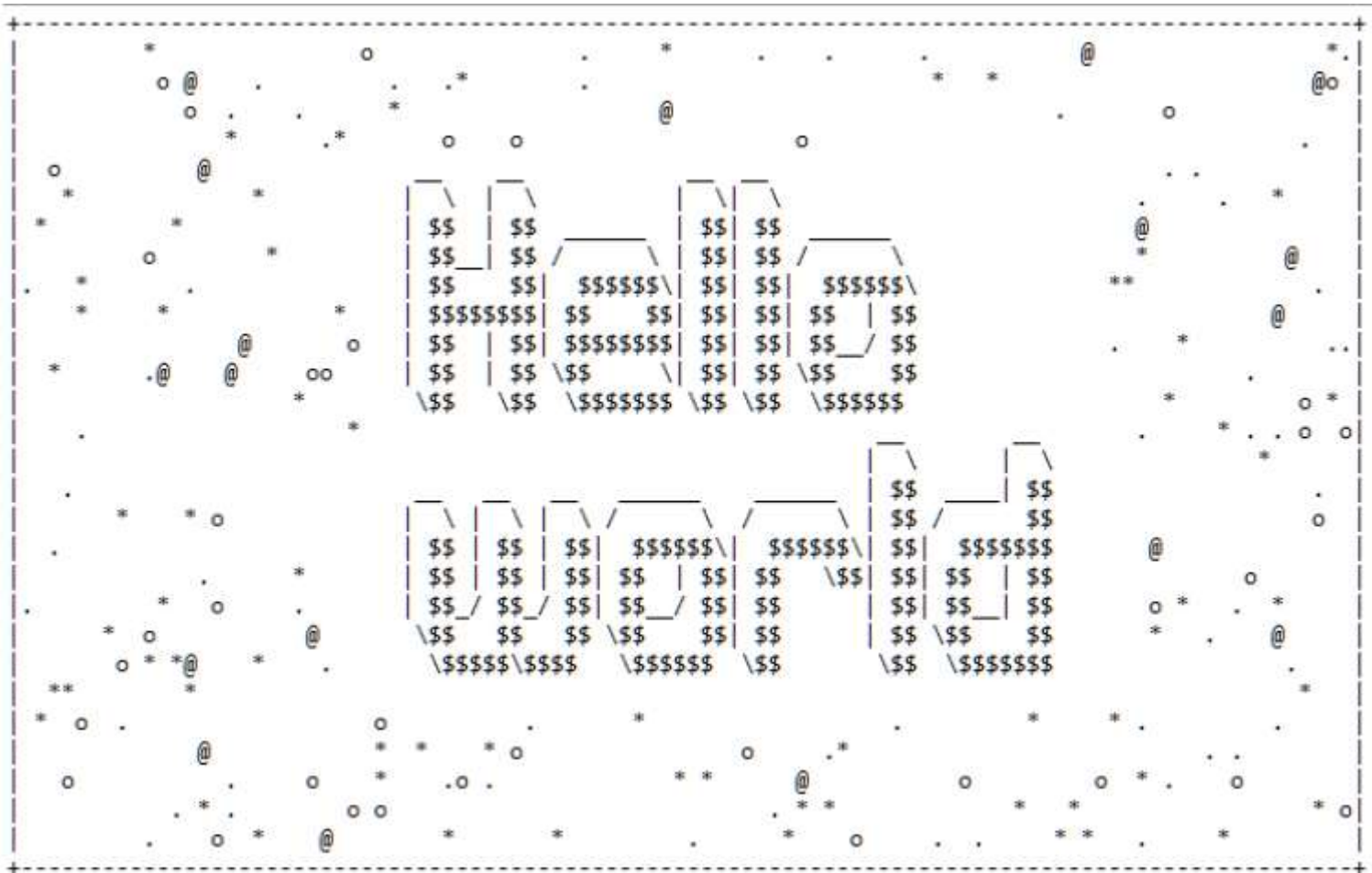




ハンズオン

ハンズオン内容

うまく動作しないプログラムをデバッガで確認して、以下のように表示されるよう修正してみてください。



解説

1.IndexError例外

方針

IndexErrorの例外をブレイクポイントに設定

原因

drawing_objectの行の長さよりもインデックスが超えてしまっているため、例外が出力される

解説

2.全行同じ表示

方針

Fieldクラスの各メソッドにブレークポイントを設置

原因

Pythonで2次元配列の静的確保をする際には二次元目を*すると参照自体をコピーしてしまうため、ある列の変更は全ての行に反映されてしまう

解説

3.アスキーアートの表示がおかしい

方針

draw_to_fieldやWordクラス等Hello worldが入れた変数を扱っている箇所を確認

原因

Hello worldのアスキーアートの中にバックスラッシュが入っているがこれが制御文字としてとらえられてしまうため、表示がおかしくなる

所感

今回デバッグの手法ではIDE毎の差異はあまり内容に感じられました。また、Pythonは処理が想定しない形で止まる時は例外をスローすることがほとんどのため、例外時のブレークポイントが有用だなと感じます。

参加ありがとうございました。

Twitter: @Euphoricwavism