

Pythonアプリケーションのデバッグ作業

デバッグ作業とは

実装したプログラムが意図通り（設計通り）に動いているかを確認した上で、エラーの発生などの問題（バグや誤り）があれば修正を行う作業のことを呼ぶ。

エラーの分類

主にエラーは下記の3つのエラーに分類できる

- 文法エラー (Syntax Errors)
- 実行エラー (Runtime Errors)
- 論理エラー (Logical Errors)
 - Name Error
 - Index Error
 - Type Error
 - Tab Error
 - ModuleNotFoundError

エラーに対峙する際の心構え

とにかくエラーメッセージを（しっかり）読むこと

※エラー内容をそのまま検索しても原因を探ることができる

とにかくエラーメッセージをしっかり読みましょう

文法エラー (Syntax Errors)

以下のサンプルコードは実行時に文法エラー (Syntax Errors) が発生する

```
if __name__ == '__main__':  
    for i in range(0, 10):  
        val = 2*i+2  
        print("val, val)
```

エラーの内容は下記の通り,

```
$ python3 sample.py  
File "sample.py", line 5  
    print("val, val)  
                ^  
SyntaxError: EOL while scanning string literal
```

エラー文の下記の行から `sample.py` の5行目の `print()` においてエラーが生じている

```
File "sample.py", line 5
    print("val, val)
```

エラーの内容としては文法エラーであることがわかる.

`string literal` の `scanning` 時に `EOL` (End Of Line)が見つかったとのこと.

```
SyntaxError: EOL while scanning string literal
```

ここまでの手がかりでエラーの発生したソースを見てみると文字列を定義するためのダブルクォーテーションが抜けている部分がある.

```
if __name__ == '__main__':

    for i in range(0, 10):
        val = 2*i+2
        print("val, val)
```

実行エラー (Runtime Errors)

以下のサンプルコードは実行時に実行エラー (Runtime Errors) が発生する

```
if __name__ == '__main__':  
    for i in range(0, 10):  
        val = 10/i  
        print("val", val)
```

エラーの内容は下記の通り,

```
$ python3 sample.py  
Traceback (most recent call last):  
  File "sample.py", line 4, in <module>  
    val = 10/i  
ZeroDivisionError: division by zero
```

エラーは `sample.py` の4行目, `val = 10/i` の部分で発生している.

```
File "sample.py", line 4, in <module>
    val = 10/i
```

エラーの内容としては `ZeroDivisionError: division by zero` との記載がある

```
ZeroDivisionError: division by zero
```

エラー内容から0除算エラーとわかる.

※for文における繰り返し時に, `i=0` となるタイミングで0除算が発生した.

論理エラー（その他の「思い通りに動かない」エラー）

Name Error

```
array = [0,1,2,3,4,5]

sum = 0
for i in range(0, len(array)):
    sum = sum + array[i]
    print(sum)

average = sam/len(array)
```

```
Traceback (most recent call last):
  File "sample.py", line 8, in <module>
    average = sam/len(array)
NameError: name 'sam' is not defined
```


IndexError

```
array = [0,1,2,3,4,5]

sum = 0
for i in range(0, 10):
    sum = sum + array[i]
    print(sum)

average = sum/len(array)
```

```
Traceback (most recent call last):
  File "sample.py", line 5, in <module>
    sum = sum + array[i]
IndexError: list index out of range
```

TypeError

```
import csv
f = open('text.csv', 'r')
dataReader = csv.reader(f)

for row in dataReader[0:2]:
    print(row)
f.close()
```

```
Traceback (most recent call last):
  File "sample.py", line 5, in <module>
    for row in dataReader[0:2]:
TypeError: '_csv.reader' object is not subscriptable
```

→ `csv.reader()` で返ってくる変数である `dataReader` はオブジェクトとして扱われているため、リスト型のようにインデックス参照できない。

しかしサンプルコードを見るに、for文で回すことができるのでリストとして参照できそう。

ということで明示的に `list()` を用いてlist型に型を指定した上で扱ってみると・・・

```
import csv
f = open('text.csv', 'r')
dataReader = csv.reader(f)
dataReader_dst = list(dataReader)

for row in dataReader_dst[0:2]:
    print(row)
f.close()

print(type(dataReader))
print(dataReader)
print(type(dataReader_dst))
print(dataReader_dst)
```

list型にするとスライスを用いた指定も可能になった

実行結果：

```
['a', 'b', 'c']  
['1', '2', '3']  
<class '_csv.reader'>  
<_csv.reader object at 0xb6a3dfb0>  
<class 'list'>  
[['a', 'b', 'c'], ['1', '2', '3'], ['1', '2', '3'], ['1', '2', '3'], ['1', '2', '3'], ['1', '2', '3']]
```

Tab Error

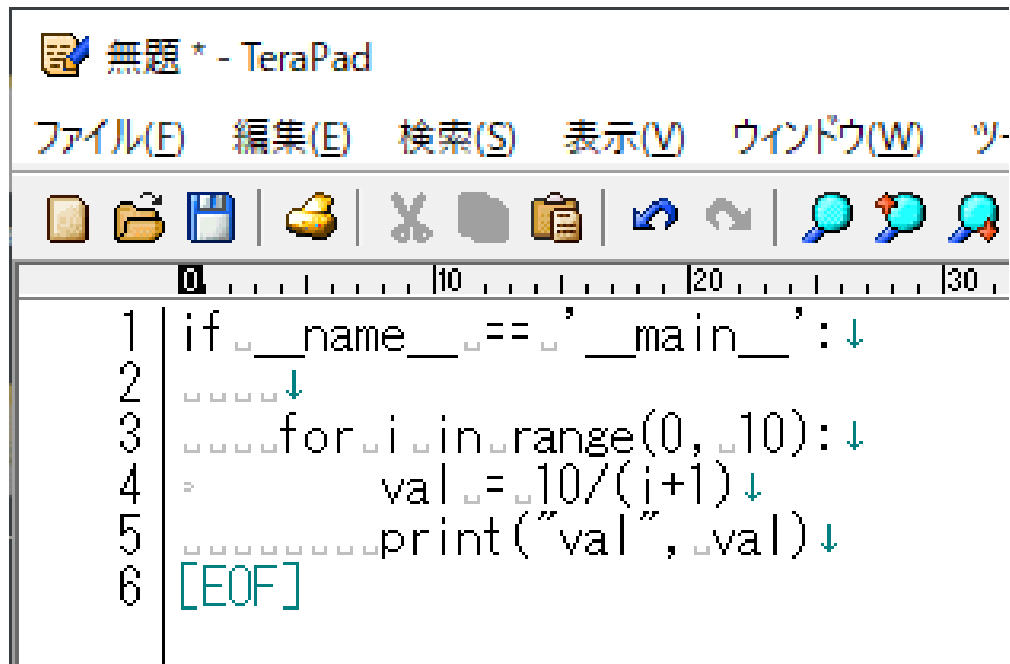
```
if __name__ == '__main__':  
    for i in range(0, 10):  
        val = 10/(i+1)  
        print("val", val)
```

```
$ python3 sample.py  
File "sample.py", line 4  
    val = 10/(i+1)  
              ^  
TabError: inconsistent use of tabs and spaces in indentation
```

実は `val = 10/(i+1)` の行のインデントがTabになっていた。

その他の行は半角スペースが4つでインデントされていたため、pythonのインタプリタが分構造を解析できずにエラーが出た。

インデントの方法が混在している場合にはTab Errorが発生するため、インデントはソースコード内で統一する。なお、スペースやTabの表示を区別する高機能なエディタを利用することで事前に回避できる。



The screenshot shows a TeraPad text editor window titled "無題* - TeraPad". The menu bar includes "ファイル(F)", "編集(E)", "検索(S)", "表示(V)", "ウィンドウ(W)", and "ツール(T)". The toolbar contains icons for file operations (new, open, save, print, delete, copy, paste) and editing (undo, redo, find, replace). The code editor displays a Python script with line numbers 1 through 6. Line 1 is `if __name__ == '__main__':`. Line 2 is `↓`. Line 3 is `for i in range(0, 10):`. Line 4 is `val = 10/(i+1)`. Line 5 is `print("val", val)`. Line 6 is `[EOF]`. The code is indented with spaces, but the line containing `val = 10/(i+1)` is indented with a tab character, which causes a Tab Error when the script is executed.

```
1 if __name__ == '__main__':  
2     ↓  
3     for i in range(0, 10):  
4         val = 10/(i+1)  
5         print("val", val)  
6 [EOF]
```

ModuleNotFoundError

```
import pandas as pd

df = pd.read_csv("hoge.csv")
```

```
$ python3 sample.py
Traceback (most recent call last):
  File "/home/pi/sample/sample.py", line 1, in <module>
    import pandas as pd
ModuleNotFoundError: No module named 'pandas'
```

モジュールがインストールされていない、または名前が間違っている、重複した変数を定義したなどで参照できないことが原因。

モジュールがインストールされていない場合は `pip` コマンドなどでインストールする。

デバッグの活用

VSCodeなどの機能拡張によって、デバッグを利用できる。
デバッグが利用できるとプログラムを走らせながらコード内の変数の移り変わりなどを確認することができる。

| [VS Codeを使ってPythonコードをデバッグするための基礎知識](#)”

| [Visual Studio Code でPythonファイルをデバッグする方法](#)”

（複雑なプログラムのデバッグ作業やモジュールのバグなどを見つけた際など）大規模なプログラムの場合にはデバッグの強力なツールになる

→ バグが出ないようにするには、全体の見通しの効く小さな機能の単位でよくデバッグすること。