

13 Python基礎

エラーと例外

- P.85～90までが基本的な例外処理。良く読んでください。
 - クラスについては、後に説明するので、例外処理の基本的な使い方を理解してください。

<https://note.nkmk.me/python-try-except-else-finally/>

エラーと例外の基本概念

エラーと例外の違い

- エラー (Error)
 - プログラムの実行を継続できない重大な問題
 - 例：構文エラー (SyntaxError)、インデントエラー (IndentationError)
 - プログラムの修正が必要で、実行時には解決できない
- 例外 (Exception)
 - プログラムの実行中に発生する予期せぬ状況
 - 例：ファイルが見つからない (FileNotFoundError)、ゼロ除算 (ZeroDivisionError)
 - 適切な処理をすることで、プログラムの実行を継続できる可能性がある

なぜ例外処理が必要なのか？

1. プログラムの堅牢性向上

- 予期せぬ状況でもプログラムが突然停止しない
- ユーザーに適切なエラーメッセージを表示できる
- エラー発生時のデータ損失を防ぐ

2. デバッグのしやすさ

- エラーの原因を特定しやすくなる
- エラーが発生した場所と状況を正確に把握できる
- ログ記録による問題の追跡が容易になる

3. コードの可読性向上

- エラー処理と通常の処理を分離できる
- コードの意図が明確になる
- メンテナンス性が向上する

例外処理の基本的な考え方

1. LBYL : 防御的プログラミング (Look Before You Leap)

- エラーが発生する可能性を事前に考慮する
- 適切なエラーメッセージを準備する
- エラー発生時の代替処理を用意する

2. EAFP (Easier to Ask for Forgiveness than Permission)

- 「許可を求めるより、許しを請う方が簡単」という考え方
- 先に処理を試みて、失敗した場合に例外を処理する
- 例 :

```
# 良い例 (EAFP)
try:
    value = my_dict['key']
except KeyError:
    value = 'default'

# 避けるべき例
if 'key' in my_dict:
    value = my_dict['key']
else:
    value = 'default'
```

3. 例外の伝播 (Propagation)

- 例外は発生した場所から呼び出し元へと伝播する
- 適切な場所で例外を捕捉する
- 必要に応じて例外を再送出 (re-raise) する

よく使用される例外の例

1. ValueError

- 不適切な値が渡された場合
 - 例：文字列を整数に変換できない

2. TypeError

- 不適切な型の値が渡された場合
 - 例：数値に文字列を加算しようとした

3. FileNotFoundError

- ファイルが見つからない場合
 - 例：存在しないファイルを開こうとした

4. ZeroDivisionError

- ゼロで除算しようとした場合
 - 例：数値を0で割ろうとした

5. KeyError

- 辞書に存在しないキーにアクセスした場合
 - 例：存在しないキーで辞書の値を取得しようとした

例外処理の記述

```
try:
    処理A
except 例外名 as 変数名x:
    変数名xを使用した処理・出力
except 例外名 as 変数名y:
    変数名yを使用した処理・出力
else:
    処理Aが正常時の処理
finally:
    必ず行う処理（後処理）
```

【組み込み例外のクラス階層】

<https://docs.python.org/ja/3/library/exceptions.html#exception-hierarchy>

- `except Exception` を指定した場合、それ以下の例外すべてを補足する
 - `ArithmeticError`
 - `AssertionError`
 - `AttributeError`
 - `:`
- あまり上位のクラスを指定すると、例外の補足が難しくなるので注意

例外の適切な使い方

<https://qiita.com/hasoya/items/05d4e49d492869875cca>

- EAFP(Easier to Ask for Forgiveness than Permission)の考え方は重要

<https://tasukehub.com/articles/me46svn3dzt/>

- 良くまとまっている

<https://google.github.io/styleguide/pyguide.html#s2.4-exceptions>

- 英語版しか無いが、プログラム作成のヒントが多く書かれている

吉村のお薦め本

- Python Distilled —プログラミング言語Pythonのエッセンス（オライリー・ジャパン）



- ロバストPython –クリーンで保守しやすいコードを書く（オライリー・ジャパン）



例外発生時のデバッグ方法

VSCodeでのデバッグ基本手順

1. ブレークポイントの設定

- デバッグしたい行の左側をクリックして赤い点を表示
- 例外が発生する可能性のある箇所に設定
 - 例：ファイル操作、数値計算、リスト操作など

2. デバッグの開始

- F5キーを押すか、左側のデバッグパネルから「実行とデバッグ」を選択
- 初回は「Python File」を選択

3. デバッグ中の操作

- F10: ステップオーバー（現在の行を実行）
- F11: ステップイン（関数の中に入る）
- Shift+F11: ステップアウト（関数から出る）
- F5: 続行（次のブレークポイントまで実行）

例外発生時のデバッグ

1. 例外ブレークポイントの設定

- デバッグパネルの「例外」タブを開く
- 「Python Exceptions」にチェックを入れる
- 特定の例外のみをキャッチしたい場合は、その例外にチェックを入れる

2. 変数の確認

- デバッグパネルの「変数」タブで現在の変数の値を確認
- マウスを変数に重ねると値が表示される
- ウォッチ式を追加して特定の式の値を監視可能

3. コールスタックの確認

- デバッグパネルの「コールスタック」タブで例外が発生するまでの関数呼び出しを確認
- 各呼び出しをクリックして、その時点の変数の状態を確認可能

ログの活用

1. print文によるデバッグ

```
try:
    # 処理
    print(f"変数の値: {variable}") # デバッグ用の出力
except Exception as e:
    print(f"エラー発生: {e}") # エラー情報の出力
```

2. loggingモジュールの使用

```
import logging

# ログの設定
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(levelname)s - %(message)s',
    filename='debug.log'
)

try:
    # 処理
    logging.debug(f"変数の値: {variable}")
except Exception as e:
    logging.error(f"エラー発生: {e}", exc_info=True)
```

3. VSCodeのデバッグコンソール

- デバッグ中に「デバッグコンソール」タブで変数の値を確認
- 式を直接入力して評価可能
- ログ出力がリアルタイムで表示される

デバッグのベストプラクティス

1. 段階的なデバッグ

- 大きな問題を小さな部分に分割
- 各部分でブレークポイントを設定
- 変数の値が期待通りか確認

2. 条件付きブレークポイント

- ブレークポイントを右クリック
- 「条件を編集」を選択

- 特定の条件が満たされた時のみ停止

```
# 例：変数が特定の値の時のみ停止
variable == 100
```

3. エラーメッセージの活用

- エラーメッセージをよく読む
- スタックトレースから問題の箇所を特定
- エラーの種類に応じた対処方法を検討

4. デバッグ時の注意点

- 本番環境のデータは使用しない
- 機密情報はログに出力しない
- デバッグ用のprint文は最終的に削除する

練習1

- 以下の各コードを例外処理を使用したコードに書き換えてください。

(1) ユーザーから入力された値を整数に変換し、その値が正の整数であるかどうかをチェックする。
負の値が入力された場合や整数に変換できない場合はエラーメッセージを表示する

【ren1_1.py】

```
value = input("数値を入力してください： ")

if value.isdigit():
    number = int(value)
    if number > 0:
        print("正の整数です")
    else:
        print("負の値またはゼロが入力されました")
else:
    print("整数を入力してください")
```

(2) ファイルを開いてその内容を読み取る。
ファイルが存在しない場合や読み取りに失敗した場合はエラーメッセージを表示する。

【ren1_2.py】

```
file_name = input("ファイル名を入力してください： ")

file = open(file_name, 'r')
if file:
    content = file.read()
    print(content)
    file.close()
else:
    print("ファイルを開くことができませんでした")
```

- (3) リストの要素をインデックスでアクセスする。
指定したインデックスが範囲外の場合はエラーメッセージを表示する。

【ren1_3.py】

```
numbers = [10, 20, 30, 40, 50]
index = int(input("インデックスを入力してください： "))

if 0 <= index < len(numbers):
    print(f"リストの要素： {numbers[index]}")
else:
    print("インデックスが範囲外です")
```

- (4) ユーザーから2つの数値を入力させて割り算を行なう。
ゼロで割り算をしようとした場合はエラーメッセージを表示する。

【ren1_4.py】

```
num1 = int(input("第1の数値を入力してください： "))
num2 = int(input("第2の数値を入力してください： "))

if num2 != 0:
    result = num1 / num2
    print(f"結果： {result}")
else:
    print("ゼロで割り算はできません")
```

- (5) 辞書からキーで値を取得する。
指定したキーが存在しない場合はエラーメッセージを表示する。

【ren1_5.py】

```
data = {"apple": 100, "banana": 200, "cherry": 300}
key = input("キーを入力してください: ")

if key in data:
    print(f"値: {data[key]}")
else:
    print("指定したキーは存在しません")
```

例外を投げる

- プログラムが強制的に例外を発生させる

```
raise 例外クラスのインスタンス
```

※ 「レイズ」と読む

- 先の練習問題は、`try except` を使用せずに、`raise` を使用して記述することもできる。
 - 試しに2つほど選んで、書き換えてみよう。