例外処理

例外とは

- 構文が正確であるにもかかわらず、`実行時に発生する`エラーを言う

ValueError と ZeroDivisionError のコード

- ValueError
  - float型になる文字列を、int型に変換しようとしている (以下のコード例)
- ZeroDivisionError
  - 0を割ることは出来るが、`0で割ることは出来ない`

```
a = int(input("整数a："))
b = int(input("整数b："))

print(f"a * b：{a * b}")
print(f"a / b：{a / b}")

"""
- 実行例１：
整数a：12
整数b：5
a * b：60
a / b：2.4

- 実行例２：
整数a：3
整数b：1.23  # ValueError: invalid literal for int() with base 10: '1.23'

- 実行例３：
整数a：5
整数b：0
a * b：0
ZeroDivisionError: division by zero  # 5 / 0
"""
```

try　except　else　finally

1. try節

    - 例外(エラー)が起こり得る部分
2. except節
    - 例外に対する対処をする
        - 対処できない場合もあり得る (Exceptionの親クラスの場合など：BaseException)
    - 複数書くことが出来る（ただし、`最初に捕まえた部分しか実行しない`）
    - 子供クラスの例外にも対処する（is関係）
    - 複数の例外を`タプルでキャッチ`できる：後述に例文コードあり
    - Exceptionクラス以下の子供クラスを全てキャッチして、
      変数で受け取ることが出来る(`e`が良く慣習として使われている[eである必要はないが])
        - `except Exception as e:` ★ この`e`のスコープはその部分の`except節のみ`である
            - さらに`type(e).__name__`で`クラス名`を知ることが出来る
3. else節
    - except節で例外が起きなかった時、実行されます
4. fianlly節
    - 必ず実行される

- try節以外は、省略可能ではあるが、except節 または finally節 のどちらかは必須である
    - `try-except`, `try-finally`
        - try(open関数)➡ finally(close関数) などがあり得る（リソースの解放）
            - エラーが起きても、リソースを開放したい
            - しかしそれに関しては、現代では、with文を使うのが推奨されている

```
# try, except, else, finally

try:
    a = int(input("整数a："))
    b = int(input("整数b："))

    print(f"a * b は {a * b} です")
    print(f"a / b は {a / b} です")

# except節は、if文のelifのように、当てはまる最初の一つだけに対応する
except ZeroDivisionError:
    print("ZeroDivisionError発生！")
except Exception as e:
    print(f"{type(e).__name__}が発生しました")
    """
    type(〜).__name__：使い方（クラス名を知ることが出来る）：

    x = 10
    print(type(x).__name__)  # 出力: 'int'

    y = "こんにちは"
    print(type(y).__name__)  # 出力: 'str'

    z = [1, 2, 3]
```

```
    print(type(z).__name__)  # 出力: 'list'
    """
else:
    print("正常終了！")
finally:
    print("プログラムを終了します")

"""
- 例1：
整数a：10
整数b：2
a * b は 20 です
a / b は 5.0 です
正常終了！
プログラムを終了します

- 例2：
整数a：8
整数b：0
a * b は 0 です
ZeroDivisionError発生！
プログラムを終了します

- 例3：
整数a：2
整数b：3.4
ValueErrorが発生しました
プログラムを終了します
"""
```

```
# except節において、タプルで例外をキャッチする例

try:
    a = int(input("整数a："))
    b = int(input("整数b："))

    print(f"a * b は {a * b} です")
    print(f"a / b は {a / b} です")

except (ValueError, ZeroDivisionError) as e: # タプルでキャッチできる
    print(f"{type(e).__name__}が発生しました")

"""
- 例1：
整数a：3
整数b：1
a * b は 3 です
a / b は 3.0 です

- 例2：
整数a：4
```

```
整数b：5.5
ValueErrorが発生しました

- 例３：
整数a：100
整数b：0
a * b は 0 です
ZeroDivisionErrorが発生しました
"""
```

覚えておきたいError一覧

- NameError
  - 指定された名前の変数がない時
    - 例：

```
a, b = 1, 3
c = d  # NameError: 'd'は定義されていません
```

- ZeroDivisionError
  - 0は割れるが、`0`では割れない：その時のエラー
    - 例：`print(3 / 0)`
- IndexError
  - インデックスが、範囲に無い時
    - 例：`print([1, 2, 3][3])`：indexは`2`までしかない
- KeyError
  - 指定したキーが、辞書にない時
    - 例：`print({"x": 1}["y"])
- TypeError
  - 例：`print("Hi" + 10)`（str型とint型では足し算できない）
- ValueError
  - 例：

```
print(int("1"))
print(int("one"))  # "one" は、変換できない
```

- FileNotFoundError：
  - 例：

```
"""
file.txtは存在しない
（file_1.txt は、存在していた）
"""
```

```
with open("file.txt", mode="rt") as f:
    print(f.read())
```

raise文で、例外を発生させる

- `raise エラーの名前`

```
# raise文による例外の送出

def func(number: int) -> None:
    if number == 0:
        raise ValueError
    elif number == 1:
        raise ZeroDivisionError


num = int(input('number：'))

try:
    func(num)

except Exception as e:
    print('raise文の始動です')
    print(type(e).__name__)

"""
- 例１：
number：0
raise文の始動です
ValueError

- 例２：
number：1
raise文の始動です
ZeroDivisionError

- 例３：
number：5  # 例外は発生しない
"""
```

raise を単独で使って、直前の例外を再発生させる

- コード例：

```python
# raise を単独で使って、直前の例外を再発生させる

try:
    print(5 / 0)
except ZeroDivisionError:
    print("0で割ることは出来ません")
    # Errorが発生する
    raise  # ZeroDivisionError: 0で割られている割り算です

"""
0で割ることは出来ません
Traceback (most recent call last):
  ~ error_again.py", line 4, in <module>
    print(5 / 0)
          ~~^~~
ZeroDivisionError: division by zero
"""
```

END