例外処理

プログラムにおいて、予想しない事態が起きたとき、「プログラムに通知する」、または、「プログラムの処理を止める」などの処理をさせる必要がある[1]。こうした処理はどのプログラミング言語でも「例外処理」として、予約語や構文で処理することができる。

◆例外の発生させる

・タイプの違う入力　ValueError

以下のサンプル・プログラムを作成する。

```
#1
while True:
    price = int(input('price: '))
    count = int(input('count: '))
    print('price//count=',price//count)
    print('-'*25)      ・・・「-」による 25 個の区切り線
```

上のプログラムの結果は「615」

これに、「abc」を代入してみると、当然、エラーとなる

```
ValueError: invalid literal for int() with base 10: 'abc'
```

プログラム本体で、入力を int 型に規定しているので、「浮動小数点」を入力しても同様の事が起こる。

・「0」で割るエラー　ZeroDivisionError

また、count に「0」を入力すると、0 で割ることになるので、これもエラーになる

```
ZeroDivisionError: integer division or modulo by zero
```

1　処理を強制的に終了するには「Ctrl」＋「C」を使う

■例外処理

Python で例外処理を記述するには「try」文を書きます。
try を使えば、処理の流れが分断されずに、簡単にエラー処理が書ける。

```
文法：
  try:
    文・・・
  except 例外：
    文・・・
```

try 以下の文を「try 節」、except 以下の文を「except 節」と呼ぶ。

#1 のソースに**例外処理**を加える

```
#2
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except ValueError:
        print('Input an integer!')
```

◆複数のエラーを別々に書く

```
#3
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except ValueError:
        print('Input an integer!')
    except ZeroDivisionError:
        print('The count must be !=0')
```

※この文は while であるので永遠に動き続けている。なので、1 回ごとに「停止」させる必要がある。

◆複数のエラーをまとめて書く

```
#4
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except (ValueError, ZeroDivisionError):
        print('Error!')
```

◆あらゆる例外を処理する

```
#5
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except:
        print('Error!')
```

◆Exception 指定

あらゆる例外を処理した場合、「Ctrl＋C」でも終了できなくなる場合がある。その場合には、Anacondaのターミナルごと終了する必要がある。

あらゆる例外を処理したいが、プログラムを終了できなくなるのが困る場合には、例外クラス Exception を指定する。これは例外で指定した例外処理だけではなく、指定した例外を基底クラスとする例外までも処理します。

```
#6
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except Exception:
        print('Error!')
```

◆例外のオブジェクトを受け取る

例外が発生したとき、例外クラスのオブジェクトが生成されている。このオブジェクトを except 節で受け取るとり、表示することで例外に関する情報が表示できる。

```
文法：
  except 例外 as 変数：
    文・・・
```

```
#7
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except Exception as e:
        print(e)
```

多くの言語では、この例外オブジェクトを変数「e」で受け取る

◆例外が起きても無視

例外が発生しても何も行わない。どちらかと言えば、「隠してしまう」感じ。

```
文法：
  except 例外:
    pass
```

◆例外が発生しなくても実行する

例外の発生の有無に関わらず、最後に実行したい処理を finally を使って記述できる。

```
文法：
  try:
  except 例外:
    文・・・
  finally:
    文・・・
```

```
#8
while True:
    try:
        price = int(input('price: '))
        count = int(input('count: '))
        print('price//count=',price//count)
    except Exception as e:
        print(e)
    finally:
        print('-'*25)
```

◆例外が発生しなかった場合に実行する else

else 節は、except 節より後に記述し、インデントしている限りは else 節の内側として扱われる。

```
文法：
  try:
  except 例外:
    文・・・
  else:
    文・・・
  finally:
    文・・・
```

例外が発生しなかった場合の処理を try で行うことができる。

```
文法：
  try:
    文・・・  例外が発生する可能性のある処理
    文・・・  例外が発生しなかった場合に実行する処理
  except 例外:
    文・・・
  finally:
    文・・・
```

ただし、この書き方は曖昧になるので「else」を使った方が良い。

◆もし、try-except 処理を使わないと

入力された文字を if 文でさまざまなケースを調べなければならない。そのため、ソースが無駄に長くなる。

```
#9
while True:
    price = input('price: ')
    if not price.isnumeric():
        print('Input an integer')
        continue
    count=input('count:')
    if not count.isnumeric():
        print('Input an integer')
        continue
    price = int(price)
    count = int(count)
    if count ==0:
        print('The count must be !=0')
        continue
    print('price//count=',price//count)
```

◆問題

次のプログラムの出力を考えなさい。

```
#10
try:
    print(1)
    try:
        print(2)
        1/0
        print(3)
    except ValueError:
        print(4)
    finally:
        print(5)
    print(6)
except ZeroDivisionError:
    print(7)
finally:
    print(8)
print(9)
```