目次へ

エラーで落ちないスクリプトを書く

ここでは、「例外処理」というものを説明するつもりです。簡単に言うと、エラーが発生しても簡単に終了しないための仕組みです。

エラーでスクリプトが終了するのは、すでにイヤというほど経験していることでしょう。どんなときにエラーが出たか、いくつか思い出してみましょう。

- スクリプトの文法が間違っていた
- リストに10個だけ値が入っているときに、[20]番目の値を取り出そうとした
- 辞書にないキーをつかって中身を取り出そうとした
- 数字と文字列を足し算しようとした
- int( )で文字列から数にしようとしたが、'A10'なんていう文字列だったので無理だった
- 存在しないファイルを読み込み用にopenしようとした
- 日付オブジェクトを作るのに、ありえない日付(10月32日とか)を指定した

どれもつらい思い出です。いや、大げさですが。

こういったエラーが起こらないようにするには、今からやろうとしていることが本当に大丈夫かをあらかじめ確かめておくというのもひとつの手です。思いつくのは下のような対処かな。

- スクリプトの文法は絶対に間違わないようにする
- リストのn番目を参照するときには、あらかじめリストの長さを調べておいて、変な場所を調べないように注意する
- 辞書の中身を調べるときは、必ず has_key を試してキーが存在するかを調べる
- 数字に変換しようとしている文字列に変な文字が入っていないかを完全に調べておく。マイナス記号とか小数点とかも場合によっては要チェック
- ファイルが存在することをあらかじめ調べておく。また、ファイルを開く権限がないときもあるから、それもチェックする
- 日付として矛盾した数字が入っていないか、徹底的に調べておく

後半あたりから、ちょっと無理くさい努力が必要そうになってきますね。日付として矛盾した云々なんて、自力でうるう年だの何だのを調べなきゃチェックできないよ。それを楽にしたくてdatetimeモジュールを使いたいのに、本末転倒じゃない。

ってことで、発想を転換してみましょう。「とりあえずやってみて、エラーだったらこういう対処をする」という記述をしてしまうわけ。pythonにはそんな方法が用意されています。それが「例外処理」というもの。文法的には、try と except というものを覚えます。

下の例をまず見ましょう。日付オブジェクトの生成に失敗している例です。

```
>>> import datetime
>>> d = datetime.date(1999, 4, 31)
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError: day is out of range for month
```

ValueError というメッセージに注目しましょう。これが、例外の名前です。datetimeモジュールは、dateオブジェクトを作るために適切でない数字が入れられると、ValueError という例外を発生させるという仕様なんですね。

「例外」っていう用語ってなんか変ですね。Exception っていう用語を翻訳したもののようです。エラーのことを「例外」って呼ぶんだ、くらいに理解しておけばいいでしょう。

他に、色々とムチャなことをしたときにどういう名前の例外が発生しているかを見ていきましょう。

```
>>> s = 'Hello12345'
>>> print int(s)
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
ValueError: invalid literal for int() with base 10: 'Hello12345'
```

変な文字列を無理に数字に変換すると、このときもValueErrorという例外が発生する、と。

```
>>>f = open('ultimate_truth.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 2] No such file or directory: 'ultimate_truth.txt'
```

存在しないファイルを読み込もうとすると、IOErrorという例外が発生します。

```
>>> a = [1, 2, 3, 4, 5]
>>> print a[100]
```

```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

リストが持っている範囲の外を見ようとすると、IndexError。

```
>>> d = {'name': 'yama', 'age': 37}
>>> print d['job']
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
KeyError: 'job'
```

辞書にないキーにアクセスしようとすると、KeyError。いろいろな例外があるもんだ。

例外の名前を色々と確認したのは、「今からやる処理は、ひょっとするとこの例外を発生させるかも知れない」とあらかじめ予想しておかないと、それに対処するスクリプトが書けないからです。

では、それに対処するスクリプトの例を下に示します。

```
import datetime

y = 2000
m = 8
d = 41

try:
  d = datetime.date(y, m, d)
except ValueError:
  print "not valid date. I'll use some apropriate date instead."
  d = datetime.date(1970, 1, 1)

print d
```

このスクリプトの実行結果。

```
not valid date. I'll use some apropriate date instead.
1970-01-01
```

try と except は、if と else みたいに、それぞれの終わりにコロン「:」がついて、その下に書く処理は字下げが必要です。

で、tryの下にある部分が、「エラーが起こるかもしれないけど、やってみる」ところ。 except の下は、残念ながらエラーが起こった（例外が発生した）ときに、臨時で実行されるところです。except ValueError と書いてあるから、ValueError が出たときだけ実行される臨時処理です。

だから、日付オブジェクトを作ることは一旦は失敗しましたが、これが try の中だったおかげで、except が後始末することができました。スクリプトの実行は中断されず、最後の print d の部分では、臨時処理の中でその場しのぎ用に作った「1970年1月1日」という日付を使うことができました。

try: の下には、ここでは一行分の処理しか書いてませんが、複数行の処理を順番に書いてもいいです。とにかく、try: の配下にあるスクリプトがどこかでエラーを起こした時は、残りの try: 配下を実行することを中断して、これに対応する except: を見つけてそこに実行をジャンプしてくれます。

とりあえず最初のところは、ここまで使えるようになれば充分と思います。

なんで例外の種類を予想しておかなくてはいけないかというと、実行中断すべきスクリプトは実行中断すべきだからです。例外処理ってのは強力なものですから、スクリプトの作成ミスなんかで予想外のエラーが発生したときは、やっぱり実行中断して修正しなくてはいけないでしょう。バグがあるのになぜか走り続けるスクリプトでは始末が悪いです。死んだのに死なない、ゾンビ野郎です。例外処理は、本当にやるべきときだけやりましょう。

- ここでは「必ず、発生するかも知れない例外を予想せよ」と書きましたが、実はどんな例外も受け付けるような書き方もあります。これを使うと手が抜けるんですが、あまり望ましくないのも本当なので、まずはちゃんとしたマナーで例外処理をしましょうね。（半ば、著者自身の自戒）

あ、文法エラー（例外の名前はSyntaxError）だけは、通常、例外処理はできませんので注意しましょう。文法エラーが起こってるってことは、そもそもpythonスクリプトとして実行開始できないってことですからね。例外処理どころじゃありません。スタートさえできてないですから。（もっとも、他のスクリプトからインポートするときとかは別ですが...今はまだいいですね）

文法エラーってのは下みたいなやつね。文字列の開始と終了で違うマークをつけちゃった、とかそんなのです。

```
>>> print "Hello'
  File "<stdin>", line 1
    print "Hello'
                 ^
SyntaxError: EOL while scanning string literal
```

二種類以上の例外を待ち受ける

ところで、ある処理をしようとして、二種類以上の例外が出てくるかも知れないときがあります。

たとえば、

```
print a[x] + 10
```

みたいなことをしたいとする。aはリストで、そこのx番目の値に10を足したいということがわかりますね。

起こりうるエラーその1は、リストが短くて、x番目の値が存在しないとき。このときは IndexError が起こります。その2は、リストに値はあったんだけど、中身が文字列か何かだったせいで、足し算ができないとき。このときは TypeError が起こります。片方だけのエラーを予測したスクリプトでは、やっぱり落ちる可能性が残るものになってしまいますね。

こういうときは、下のように書けます。

```
try:
  print a[x] + 10
except IndexError:
  print "IndexError Raised!"
```

```
  except TypeError:
    print "TypeError Raised!"
```

expeptの部分は、ひとつのtryに対してたくさん現れていいことになっています。だからこんな書き方ができますよ。

または、

```
  try:
    print a[x] + 10
  except (IndexError, TypeError):
    print "some error must be occured..."
```

こんな風に、どっちの例外でも同じ後始末で済むときは、カッコでくるんで、コンマで好きなだけ待ち受ける例外の種類を書き連ねてもよいです。

ついでだから、どんな例外にも反応するための書き方というのも紹介だけしておきます。一般に、手抜きの手段となりますから、注意してください。

```
  try:
    print a[x] + 10
    i.say('hello')
    o = open('babababababababa')
  except:
    print "nothing really matters."
```

except で例外の種類を書かなければよいというわけです。これで、どんなエラーが起こっても反応できるコードが書けますが...あとあと予想外のことが起こったときに困りますよ。たぶんね。

データの入力を受け付ける

例外処理が主に力を発揮するのは、予想外のデータを処理する必要があるときです。予想外なものとは何か。

予想できないもの、それは人間です。（完）

（完）はウソですすいません。なぜか決めゼリフっぽい雰囲気になってしまったのでつい書きました。言いたかったのは、人間が入力したデータを扱うときは、いろいろ予想外のことが起こると想定したほうがいいよということです。

新しい命令をひとつ紹介します。あとでこれを使って練習問題をしてもらう予定です。

raw_input という命令で、これが実行されると、プロンプト上から利用者が文字列を入力するのを待つ状態になります。例を下に挙げます。

```
>>> s = raw_input('HELLO:')
HELLO:
```

ここで一旦止まりますね。で、なんか適当に入力して、[Enter]を押しましょう。

```
>>> s = raw_input('HELLO:')
HELLO:nani ga HELLO da
```

```
>>> s
'nani ga HELLO da'
```

変数 s に今入力したデータが入りました。今は対話シェルで試していますが、スクリプトにして実行しても同じように一行入力させることができますよ。

これを使って、下のような練習問題をやってみようと思います。

練習問題07

【練習問題:07】一行入力した日付が妥当なものかを調べるスクリプトを書け。

仕様：

- ユーザーが 2010/11/3 といった感じのフォーマットで入力すると想定する。
- たとえば 3日のことを、3 と記しても 03 と記しても許すものとする。
- 入力した日付が妥当なら、'2010-11-03 は 水曜日' といったように出力する。
- 入力した日付が妥当でないなら、'誤った入力です' と出力する。
- いつまでも連続して入力できること。つまりひとつのデータを処理したら、再度入力プロンプトが出る。
- bye と入力されたときは、スクリプトの実行を終了する。

こんな感じでできますでしょうか。どうぞやってみてください。

あえてヒントが少なめですが、どんな種類の入力がどういうエラーを起こしそうだろうか、という想像をしてみることをお勧めします。

スクリプトの実行を終了するのってどうやるの？　と聞かれそうです。これは、sys.exit です。sysモジュールをインポートすると、そこで exit という命令が使えるようになります。具体的には下のように実行しますよ。

```
>>> import sys      ←これは、スクリプトの一番最初に一回だけ書けばよい
>>> sys.exit()
```

対話プロンプトが消えちゃったら、成功。こうやって、スクリプトの途中でも実行をやめることができます。