Chyby a výjimky

Petr Pošík

Katedra kybernetiky, FEL ČVUT v Praze

OI, B4B33RPH - Řešení problémů a hry, 2016

Chyba

Stav programu (podmínky), který mu brání v dosažení požadovaného výsledku.

Odkud se chyby berou?

Abychom dosáhli požadovaného výsledku,

- programátor musí program napsat,
- obvykle s využitím dalších knihoven,
- program se musí spustit v nějakém prostředí a
- uživatel musí do programu vložit vstupní data.

Kdo chyby opravuje?

- Lidi. Vývojář, administrátor, uživatel.
- K tomu potřebují informace.
- Program by jim je měl poskytnout.

Co můžete dělat jako programátor?

Testovat!

- Syntaktické chyby: kód nepůjde ani spustit.
- Chyby ve vašem kódu nebo v používaných knihovnách: unit testy.
- Nesoulad mezi předpoklady knihovních funkcí a vaším chápáním těchto předpokladů: integrační testy.

Co ale dělat, když

- váš kód dostane chybné vstupy, nebo
- je špatně nastavené prostředí?

Napsat kód tak, aby byl na tyto **výjimečné** situace připraven a aby poskytl uživateli/administrátorovi systému informace, jak chyby napravit!

Dvě hlavní metody práce s chybami

1. Chybové stavy jako návratové kódy funkcí.

2. Výjimky.

Obě metody mají svá pro i proti (a horlivé zastánce i odpůrce).

Chyby jako návratové hodnoty funkcí

Funkce vrací `True`/`False` (úspěch/neúspěch) nebo celé číslo (0 - úspěch, jiné číslo je kód chyby).

Klady

- Snazší debugging, jednodušší metoda, rychlejší.

Zápory:

- Každý volající kód se musí s návratovou hodnotou nějak vypořádat.
- Návratové kódy jsou často ignorovány.
- Nelze použít v `__init__`, v přetížených operátorech a pro funkce, které podle specifikací mají vracet něco jiného, než chybový kód.

Chyby jako výjimky

Jazyk musí obsahovat speciální podporu; v případě chyby se přeruší "normální" vykonávání programu a výjimka se propaguje až na místo, kde je ošetřena.

Klady:

- Flexibilita
- Kód může být jednodušší. O výjimky se nemusím starat všude, jen tam, kde vím, jak s ní naložit.

Zápory:

- Efektivita: podpora výjimek něco stojí.
- Složitější debugování - alternativní řídicí struktura v kódu.

Ošetření chyb v Pythonu

```
>>> import this
The Zen of Python, by Tim Peters

[...]
Errors should never pass silently.
Unless explicitly silenced.
[...]
```

- Python preferuje výjimky! (Nepoužívá chybové návratové kódy.)
- Jsou všude. Vestavěné do mnoha funkcí a protokolů.

Co je výjimka v Pythonu?

- Objekt (odvozený od `BaseException`).
- Obvykle indikuje, že je něco špatně. (Ale ne všechny výjimky reprezentují chyby.)
- Výjimky, které nebyly nikde ošetřeny, vybublají až k uživateli do (Python) shellu.

Příklad: Syntaktická chyba

In [1]:

```
print'How are you')
```

```
  File "<ipython-input-1-9c0f473cfb8a>", line 1
    print'How are you')
                      ^
SyntaxError: invalid syntax
```

Příklad: porušení předpokladů o datech

In [2]:

```
a = []
a[5]     # Předpokládá, že prvek na indexu 5 existuje.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-2-3c4cc7d0e911> in <module>()
      1 a = []
----> 2 a[5]     # Předpokládá, že prvek na indexu 5 existuje.

IndexError: list index out of range
```

Příklad: porušení předpokladů o datech

In [3]:

```
tup = (1, 'a', True)
tup[1] = 'b'    # Předpokládá, že prvek na indexu 1 lze změnit
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-ab4e3b3964aa> in <module>()
      1 tup = (1, 'a', True)
----> 2 tup[1] = 'b'    # Předpokládá, že prvek na indexu 1 lze změnit

TypeError: 'tuple' object does not support item assignment
```

Příklad: porušení předpokladů o parametru funkce

In [4]:

```
def square_num(number):
    """Return the square of a number"""
    return number * number     # Předpokládá, že number je číslo
```

In [5]:

```
print(square_num(3))
```

```
9
```

In [6]:

```
print(square_num('number'))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-ef60bf9dda2e> in <module>()
----> 1 print(square_num('number'))

<ipython-input-4-3d3f318576bf> in square_num(number)
      1 def square_num(number):
      2     """Return the square of a number"""
----> 3     return number * number     # Předpokládá, že number je číslo

TypeError: can't multiply sequence by non-int of type 'str'
```

Výchozí chybová hláška

Dává vývojáři mnoho informací o tom, co se skutečně stalo:

- traceback (kde nastala chyba přes všechny úrovně zásobníku)
- typ chyby (třída)
- chybová zpráva

Info, které potřebuje vývojář (k opravě kódu) **není stejné** jako info, které potřebuje uživatel/admin (k opravě vstupních dat/prostředí)!

- Nikdy neukazujte traceback uživateli!
  - K ničemu mu není.
  - Archivujte ho, pošlete si ho emailem, ...
- Chybové hlášky pro uživatele vytvářejte pečlivě!
  - Zaměřte se na to, co uživatel může udělat.
- Pokud je to vhodné, restartujte program.

Indikace chybového stavu vyhozením výjimky

Definujme jednoduchou funkci, která kontroluje přípustnost zadané hodnoty.

In [7]:

```
def ask_for_age():
    age = int(input('Enter your age:'))
    if age < 0:
        raise ValueError('{} is not valid age'.format(age))
    return age
```

Pokud je hodnota v pořádku, neděje se nic zvláštního:

In [8]:

```
ask_for_age()
```

```
Enter your age:20
```

Out[8]:

```
20
```

Pokud je zadána nesmyslná hodnota, vyhodí se výjimka (protože jsme se o to postarali):

In [9]:

```
ask_for_age()
```

```
Enter your age:-3

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-38a44bff3cde> in <module>()
----> 1 ask_for_age()

<ipython-input-7-87c4808698fe> in ask_for_age()
      2     age = int(input('Enter your age:'))
      3     if age < 0:
----> 4         raise ValueError('{} is not valid age'.format(age))
      5     return age

ValueError: -3 is not valid age
```

Pokud je zadána hodnota, která se nedá převést na číslo, postará se o vyhození výjimky Python ve funkci `int()`:

In [10]:

```
ask_for_age()
```

```
Enter your age:your age

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-38a44bff3cde> in <module>()
----> 1 ask_for_age()

<ipython-input-7-87c4808698fe> in ask_for_age()
      1 def ask_for_age():
----> 2     age = int(input('Enter your age:'))
      3     if age < 0:
      4         raise ValueError('{} is not valid age'.format(age))
      5     return age

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

Tyto výjimky by ovšem uživatel neměl vidět, jsou to informace pro nás!

Zpracování výjimek

Ke zpracování výjimek se v Pythonu používá příkaz `try-except-else-finally`:

```
try:
    # Protected code which may raise an exception
except <ExceptionType> as ex:
    # Handle exception ex
else:
    # Code which executes if no exception is raised
finally:
    # Code which executes in any case
```

Vytvořme lepší funkci pro zadání věku, která uživateli příp. sdělí, co je špatně, a zeptá se jej znovu:

In [11]:

```
def ask_for_age_better():
    while True:
        try:
            age = ask_for_age()
        except ValueError as ex:
            print(ex)
        else:
             break
    return age
```

In [12]:

```
ask_for_age_better()
```

```
Enter your age:your age
invalid literal for int() with base 10: 'your age'
Enter your age:-3
-3 is not valid age
Enter your age:20
```

Out[12]:

```
20
```