Výjimky

- Chyby jsou buďto syntaktické, nebo výjimky.
- Ty druhé dosud program také zbořily, ale dají se chytat (nebo vyvolat).
- Klíčová slova `try`, `except` a `raise`.

Výjimky

a jejich pravidla

- `try:` zahajuje blok, kde může padnout výjimka,
- `except typ_vyjimky:` zahajuje blok výjimku ošetřující
- Chceme-li ošetřit více typů výjimek, uděláme tuple (tedy seznam dáme do kulatých závorek),
- `except:` – pochytej všechny možné výjimky,
- `raise typ(parametry)` vyvolá výjimku,
- `finally` uvádí finalizátor (kód, který se vykoná jak po výjimce tak bez ní). Užitečné nechceme-li výjimku ošetřit (pošleme ji dál), ale finalizátor chceme každopádně provést.

První úloha z ReCodExu

vyřešena ošklivým způsobem

```
try:
    print(int(input())//int(input()))
except:
    print("NELZE")
```

- Výjimky bývají pomalé $\Rightarrow$ nepoužívat zbytečně,
- tento způsob použití je krajně nekultivovaný,
- vylepšení: `except ZeroDivisionError:`
- ale správné řešení mělo samozřejmě vypadat jinak.

Balíčky

jsou dalším prostředkem dělení programu

- Kód umíme dělit do funkcí, souborů, modulů.
- Moduly umíme importovat (import jmeno).
- Balíček může sestávat z několika souborů (může být obsažnější).
- V adresáři vytvoříme soubor `__init__.py` (klidně prázdný). Ten je inicializátorem balíčku.
- V něm můžeme definovat některé položky a také co se nahraje, řekneme-li `from balicek import *`:
- `__all__=[ "pokus", "druhy_pokus"]` – v tomto případě importujeme tyto dvě položky. Ostatní ne.

Příklad

balíčku p

__init__.py:

```
__all__=[ "pokus", "druhy_pokus"]
```

pokus.py:

```
def f():
    print("Nazdarek")
```

druhy_pokus.py:

```
def g():
    print("Funkce g")
```

treti_pokus.py:

```
def skrytejsi_funkce():
    print("Sem se dostat da trochu praci.")
```

Příklad

balíčku p

program.py

```
from p import *
pokus.f()
druhy_pokus.g()
#treti_pokus.skrytejsi_funkce() by spadlo
import p.treti_pokus
p.treti_pokus.skrytejsi_funkce()
# Ted uz to projde.
```

Některé rituály

a jejich smysl a význam

- `#!/usr/bin/python3` – bývá k vidění v mých zdrojácích,
- říká, který interpret má kód zinterpretovat.
- ```
  if __name__=="__main__":
      #zde nasleduje hlavni program
  ```
- Kód se provede pouze pokud nejsme importování jako modul či balíček, ale jsme hlavní program.
- Proměnná `__name__` říká naše jméno. Pokud nás nikdo neimportoval, obsahuje řetězec `__main__`.
- Možné využití je k testování knihovny. Do této sekce dáme testy jednotlivých funkcí (tzv. unit testy). Knihovnu k otestování samostatně spustíme. Testový kód se však nespustí, pokud někdo knihovnu importuje.
  ```

Klíčové slovo `assert`

se hodí ke tvorbě unit testů

- Syntax: assert podminka, "chybove_hlaseni",
- `assert` ve všech jazycích dohlíží na platnost invariantů (tedy stále platných tvrzení),
- typicky potřebujeme zkontrolovat, zda fungujeme (má smysl pokračovat).

Příklad

testy řízeného programování, unit testů a assertu

```
def secti_abs_hod(a,b):
    if(a<0):
        return b-a
    else: return a+b

if __name__=="__main__":
    assert secti_abs_hod(1,0)==1,"Chyba 1, 0"
    assert secti_abs_hod(1,-1)==2,"Chyba 1, -1"
    assert secti_abs_hod(1,1)==1,"Chyba 1, 1"
    assert secti_abs_hod(-1,1)==1,"Chyba -1, 1"
```

Testy řízené programování

je vhodné se naučit při práci s ReCodExem

- Než začneme programovat, navrhneme vhodné testy.
- Podumáme, co může být v zadání záludné,
- navrhneme testy na jednotlivé záludné případy,
- napíšeme kód,
- provedeme testy a hned uvidíme, proč nám ReCodEx nechce dát plný počet bodů.
- Toto jsou jednotkové (alias unit-)testy.

Integrační testy

tvoří další typ testování

- Zkoumají, zda moduly správně spolupracují.
- Lze je provést alternativní reprezentací (napíšeme si neefektivní bezchybnou reprezentaci a vyzkoušíme, zda software reaguje stejně v obou případech).
- Aby se snáze prováděly, mívají datové struktury standardizované operace (member, insert, delete,...), na které lze nasadit unit-testy.
- Integrační testy se týkají především sesazování jednotlivých součástí softwaru. Byl-li interface (pro tyto součásti) navržen špatně, lze očekávat velký malér.