Algoritmy a programování

Triky

Vojtěch Vonásek

Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague

Try

- “Bezpečné” vykonání části programu, aniž by program skončil chybou
- Pokud nastane chyba běhu, je zachycena a vznikne tzv. výjimka (exception)
- Dle typu výjimky lze zjistit, co se stalo

Příklad: runtime chyba

```
1 a = [1,2,3]
2 last = a[4]
3 print("Last", last)
```

```
    last = a[4]
IndexError: list index out of range
```

Try

- “Bezpečné” vykonání části programu, aniž by program skončil chybou
- Pokud nastane chyba běhu, je zachycena a vznikne tzv. výjimka (exception)
- Dle typu výjimky lze zjistit, co se stalo

Příklad: runtime chyba je zachycena v try

```
1 a = [1,2,3]
2 try:
3     last = a[4]
4 except:
5     last = None
6     print("Cannot␣acess␣element␣a[4]")
7 print("Last", last)
```

```
Cannot acess element a[4]
Last None
```

Try

- “Bezpečné” vykonání části programu, aniž by program skončil chybou
- Pokud nastane chyba běhu, je zachycena a vznikne tzv. výjimka (exception)
- Dle typu výjimky lze zjistit, co se stalo

Příklad: runtime chyba

```
1 d = {}
2 d["one"] = 1
3 d["two"] = 2
4
5 for key in ["one", "two", "three"]:
6     d[key] += 1
7 print(d)
```

```
    d[key] += 1
KeyError: 'three'
```

Try

- “Bezpečné” vykonání části programu, aniž by program skončil chybou
- Pokud nastane chyba běhu, je zachycena a vznikne tzv. výjimka (exception)
- Dle typu výjimky lze zjistit, co se stalo

Příklad: runtime chyba je zachycena v try

```
1  d = {}
2  d["one"] = 1
3  d["two"] = 2
4
5  for key in ["one", "two", "three"]:
6      try:
7          d[key] += 1
8      except:
9          d[key] = 1
10 print(d)
```

```
{'one': 2, 'two': 3, 'three': 1}
```

Filosofie práce s chybami

EAFP: “it’s easier to ask for forgiveness than permission”

- Programujeme tak, že rovnou provedeme zamýšlenou operaci
- A pokud ne, zachytíme výjímku (v bloku try/except)
- EAFP je filosofie podporovaná vývojáři Pythonu
- Časté používání try/except

```
1 d = {"p":0, "y":2, "t":3}
2
3 value = 0
4 for letter in "Python":
5     try:
6         value += d[letter]
7     except:
8         pass
9 print(value)
```

```
5
```

LBYL: “Look before you leap”

- Před každou kritickou operací provedeme test na možný vznik chyby
- Operaci provedeme pouze pokud jsme si jisti, že nenastane chyba
- Časté používání podmínek

```
1 d = {"p":0, "y":2, "t":3}
2
3 value = 0
4 for letter in "Python":
5     if letter in d:
6         value += d[letter]
7 print(value)
```

```
5
```

LBYL přístup

LBYL: “Look before you leap”

- Při psaní programu je třeba analyzovat program, predikovat možné problémy
- Je třeba navrhnout vhodné testy (podmínky)
- Vyžaduje hlubší znalosti programování než v přístupu EAFP
- Špatný návrh testů $\Rightarrow$ chyba programu

Kdy používat:

- Pokud jsme schopni zaručit správnost podmínek testování
- Pokud jsme schopni predikovat všechny možné chování programu
- V případech, kde k výjimečným situacím dochází často

EAFP přístup

EAFP: “it’s easier to ask for forgiveness than permission”

- Nevyžaduje velkou znalost programování
- Nevyžaduje analýzu programu a možných chyb
- Jednodušší na vývoj

Kdy používat:

- Pokud nejsme schopni zaručit, že test zachytí všechny možné chyby
  - Volání cizích programů
  - Práce s externímy zařízeními (USB, ethernet, sensory. . .)
  - Práce se soubory
- Ideálně tam, kde chyba nastává výjimečně

Úskalí EAFP:

- Nadužívané (hlavně) začátečníky, nevede na hlubší porozumění programování
- Přináší pocit dobře vykonané práce, protože program nepadá
- Zakrývá chyby, které by jinak byly odhalitelné
- Může zpomalovat výpočet