



Programovanie v jazyku Python

Testovanie, ladenie, výnimky a chyby
prednáška 4

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar

Testovanie programov

- **proces**, ktorého cieľom je nájsť problémy a chyby v kóde
- zatiaľ nehľadáme príčinu chýb, ani možné riešenia
- testovanie sa delí na
 - verifikáciu
 - kontroluje, či program, alebo jeho časť, splní návrhové požiadavky (požiadavky zo strany programátora)
 - nezaoberá sa iba tým, čo daný program robí, ale či to robí tak, ako chceme
 - validáciu
 - kontroluje, či program, alebo jeho časť, splní požiadavky používateľa - čo program robí
 - cieľom je zvýšiť spoľahlivosť kódu - nikdy nedosiahne 100%

Typy testovania

- unit testy
 - nezávislé testovanie prvkov kódu
 - testujú sa funkcie samostatne
- integration testy
 - testuje sa program ako celok
 - slúži na odhalenie problémov v komunikácii medzi modulmi a prvkami kódu
- vždy je najlepšie začať s unit testami

Návrh testov

- testovať program na každom možnom vstupe - nepraktické a niekedy nemožné
- testovanie na vhodne zvolenej sade testovacích prípadov
 - sada musí byť dostatočne malá, aby sme test vedeli vykonať v rozumnom časovom intervale
 - sada musí byť dostatočne veľká, aby reprezentovala všetky možné vstupy
- testovacia sada zvyčajne obsahuje
 - príklady najčastejších vstupov (očakávaný vstup)
 - príklady hraničných vstupov (extrémy)
 - príklady neplatných vstupov (asi najdôležitejšie)

Testovanie v Pythone

- zvyčajne v osobitnom súbore
- načítajte testované funkcie z primárneho súboru
- kľúčové slovo **assert** s podmienkou
- vyhodí chybu **AssertionError** ak test neprejde

```
def is_even(number):  
    return not number % 2
```

```
assert is_even(7) == False
```

- modul unittest v Pythone pre automatizáciu testovania

Ladenie

- našli sme chybu v kóde, potrebujeme ju odstrániť
- proces, počas ktorého našim cieľom je odstrániť **všetky** buggy z kódu
- chyba sa nemusí preukázať nefunkčnosťou programu
- niekedy je počas ladenia naším cieľom optimalizovať beh programu, resp. zvýšiť výkon

Prečo sa bug volá bug?

9/9


0800 Antan started
1000 " stopped - antan ✓

1300 (033) MP-MC { 1.2700 9.032847025
2.130476415 (2) 9.037846795 correct
2.130476415
2.130476415
correct 2.130476415

Relays 6-2 in 033 failed special speed test
in relay " 11.000 test.

Relays changed

1100 Started Cosine Tape (Sine check)
1525 Started Multi-Adder Test.

1545  Relay #70 Panel F
(moth) in relay.

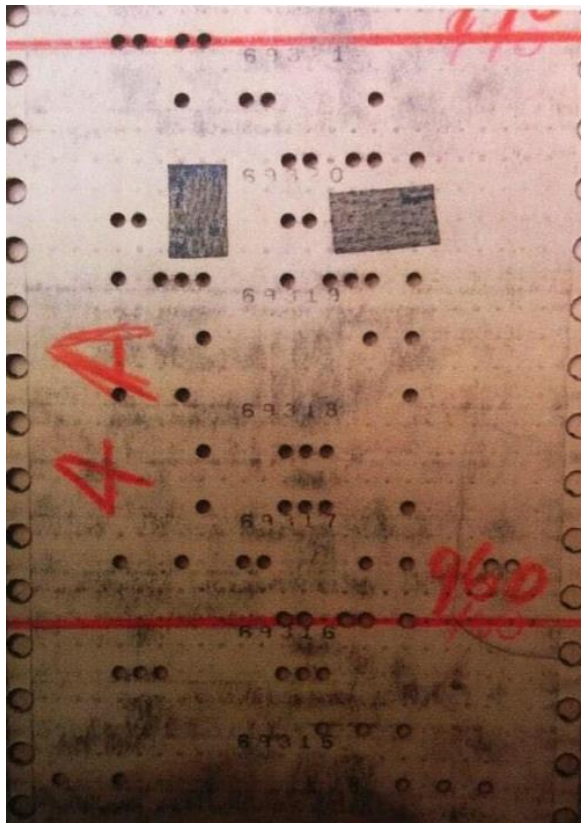
First actual case of bug being found.

1630 Antan started.
1700 closed down.

Relay
2145
Relay 3370



Prečo sa patch volá patch?



Ďalšie mýty o bugoch

1. bug sa zjaví v kóde
 - bug je v kóde preto lebo programátor ho tam (nechtiac) pridal
 - bug nie je nič viac ako omyl programátora
2. buggy sa rozmnožujú - ak človek opraví jeden bug, nahradia ho dva
 - ak zrazu musíte vyriešiť viac bugov, znamená to že ste urobili viac chýb ako ste si mysleli
 - cieľom nikdy nie je opraviť jeden bug, ale z nefunkčného kódu vytvoriť funkčný kód

Nástroje debugovania

- existuje mnoho nástrojov pre rôzne IDE
- najlepší spôsob debugovania je
 - prečítať si kód
 - operácia print

Postup při ladení kódu

How to actually learn any new programming concept



Essential

Changing Stuff and
Seeing What Happens

Postup pri ladení kódu

1. Kde sa nachádza chyba?
 2. Ako môže kód vyprodukovať chybný výstup?
 3. Je to ojedinelý problém alebo urobili sme tú istú chybu viackrát v programe?
 4. Ako opravíme chybu?
- vždy testujte na najjednoduchšom možnom vstupe
 - pri identifikácii miesta chyby znižujte priestor vyhľadávania pomocou binárneho vyhľadávania

Príklad

```
def is_palindrome():
    original_list = list()
    done = False
    while not done:
        elem = input("Enter element. Return when done. ")
        if elem == '':
            done = True
        else:
            original_list.append(elem)

    test_list = original_list
    test_list.reverse()
    return test_list == original_list
```

Typické chyby

- nesprávne poradie parametrov/argumentov
 - pravopis, preklepy, veľké/malé písmená
 - inicializácia - chceme to robiť v cykle alebo mimo?
 - rovnosť objektov vs. rovnosť hodnôt
 - aliasy vs. kópie, deep vs. shallow kópie
 - vedľajšie účinky volaných funkcií
- + každý programátor má svoje často sa opakujúce chyby

Dobré zvyky pri ladení

- skontrolovať, či testujete ten kód a súbor, ktorý upravujete
- systematická kontrola
- zapisovať čo sme už vyskúšali
- skontrolovať, či máme správne predpoklady
- ladiť kód a nie komentáre
- poprosiť o pomoc
- vysvetliť niekomu, čo program robí, resp. čo by mal robiť
- dať si pauzu
- zjednodušiť kód
- uložiť staré verzie kódu

Výnimky a chyby

- poukazují na nesprávnost programu
- dva základné typy
 - syntaktické (chyby)
 - run-time (výnimky)

Syntaktické chyby

- interpret ich identifikuje ešte pred spustením kódu
- riešia sa veľmi jednoducho
- chyba sa nie vždy nachádza tam, kde to ukazuje interpret

```
File "C:\Users\Ian\Desktop\test.py", line 2
    if condition print("okay")
                   ^
```

```
SyntaxError: invalid syntax
```

Run-time chyby/výnimky

- vznikajú počas behu programu
- zvyčajne ich spôsobujú iba vybrané vstupy

```
number = 10  
divide_by = 0  
print(number / divide_by)
```

```
File "C:\Users\Ian\Desktop\test.py", line 3, in <module>  
    print(number / divide_by)  
ZeroDivisionError: division by zero
```

Spracovanie chýb a výnimiek

- dvojica kľúčových slov `try` / `except`

```
number = 10
divide_by = 0
try:
    print(number / divide_by)
except ZeroDivisionError:
    print("You're trying to divide by zero")
```

Spracovanie viacerých chýb

```
number = 10
divide_by = 0
try:
    print(number / divide_by)
except (ZeroDivisionError, NameError, ValueError):
    print("You're trying to divide by zero")
```

Spracovanie viacerých chýb

```
number = 10
divide_by = 0
try:
    print(number / divide_by)
except ZeroDivisionError:
    print("You're trying to divide by zero")
except NameError:
    print("Undefined variable")
except ValueError:
    print("Invalid value")
```

Vyhodenie chýb

- kľúčové slovo `raise`
- umožňuje zadať ľubovoľnú chybovú správu
- ukončí vykonávanie programu ak chyba nie je spracovaná inde*

```
number = 10
divide_by = 0
if divide_by == 0:
    raise ZeroDivisionError("You're trying to divide by
zero")
print(number / divide_by)
```

Vykonanie kódu iba v prípade bez chýb

```
number = 10
divide_by = 0
try:
    print(number / divide_by)
except ZeroDivisionError:
    print("You're trying to divide by zero")
else:
    print("Everything went well")
```

Finally část'

- kód sa vykoná bez ohľadu na to, či vznikla chyba

```
try:  
    raise ValueError  
finally:  
    print("I still get printed")
```

I still get printed

Traceback (most recent call last):

File "C:\Users\Ian\Desktop\test.py", line 2, in <module>

raise ValueError

ValueError

Definícia vlastných chýb

- výnimky a chyby sú podtriedy `Exception`

```
class MyError(Exception):  
    def __init__(self, expression, message):  
        self.expression = expression  
        self.message = message
```

Práce s výnimkami

```
my_dict = dict()
if 'a' not in myDict:
    my_dict['a'] = 1
else:
    my_dict['a'] += 1

print(my_dict)
```

```
my_dict = dict()
try:
    my_dict['a'] += 1
except KeyError:
    my_dict['a'] = 1

print(my_dict)
```

Najčastejšie typy výnimiek

- `AssertionError`
- `AttributeError`
- `IndexError`
- `KeyError`
- `NameError`
- `OSError`
- `RecursionError`
- `TypeError`
- `UnicodeError`
- `ValueError`
- `ZeroDivisionError`

Zhrnutie

- testovanie programu a jeho ciele
- unit testy v Pythone
- postup pri ladení kódu
- výnimky a chyby
- práca s výnimkami v Pythone
- vybrané hlavné výnimky a ich význam