Politechnika Wrocławska

Python
wstęp do programowania dla użytkowników WCSS

Dr inż. Krzysztof Berezowski
Instytut Informatyki, Automatyki i Robotyki
Politechniki Wrocławskiej

Pytania i odpowiedzi

NAZWY ZAGNIEŻDŻONE I STRUKTURALIZACJA KODU

Lokalna przestrzeń nazw

Reguła: Jednostką przestrzeni nazw jest poziom zagnieżdżenia.

Obiekty lokalne takie jak:

- parametry funkcji,
- zmienne lokalne,
- i funkcje zagnieżdżone

są tworzone w lokalnej przestrzeni nazw i przesłaniają nazwy globalne.

Przesłanianie i dostęp do nazw globalnych

p2_functions | p1_vars

```
v1 = "Wartosc ustawiona globalnie"

def change_local():
    v1 = "100"
    print "Wartosc wyswietlona lokalnie:", v1

def change_global():
    global v1
    v1 = "Wartosc ustawiona lokalnie dla zmiennej globalnej"
    print "Wartosc wyswietlona lokalnie:", v1

print v1
change_local()
print v1
change_global()
print v1
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python4/nesting/src/p1_vars.py
Wartosc ustawiona globalnie
Wartosc wyswietlona lokalnie: 100
Wartosc ustawiona globalnie
Wartosc wyswietlona lokalnie: Wartosc ustawiona lokalnie dla zmiennej globalnej
Wartosc ustawiona lokalnie dla zmiennej globalnej
```

Funkcje zagnieżdżone

```
def f():
    print "To jest globalna funkcja f"

def g():
    print "To jest funkcja g"

    def f():
        print "Funkcja lokalna f() przeslania funkcje globalna f()"
    f()

f()
g()
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python4/nesting/src/p2_functions.py
To jest globalna funkcja f
To jest funkcja g
Funkcja lokalna f() przeslania funkcje globalna f()
```

Użycie średnika do tworzenia lokalnej przestrzeni nazw

p2_functions | p1_vars | p3_semicolon

```
a = 1

if a == 1:
    a = 2
    print "Jestesmy wewnatrz zdania zlozonego"
    print "A =", a

if a == 2: a = 3; print "Jestesmy wewnatrz zdania zlozonego"; print "A =", a

if a == 3:
    a = 4
    print "Jestesmy wewnatrz zdania zlozonego"; print "A =", a
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python4/nesting/src/p3_semicolon.py
Jestesmy wewnatrz zdania zlozonego
A = 2
Jestesmy wewnatrz zdania zlozonego
A = 3
Jestesmy wewnatrz zdania zlozonego
A = 4
```

Konstrukcje złożone

OBSŁUGA WYJĄTKÓW

Sytuacje wyjątkowe w Pythonie

Python obsługuje błędy i sytuacje wyjątkowe za pomocą **wyjątków**.

Wyjątki to dowolne obiekty „rzucane”

w poprzek hierarchii wywołań funkcji

aby zaraportować o nieoczekiwanej sytuacji

Posługiwanie się wyjątkami związane jest z

- umiejętnością przechwycenia wyjątku
- umiejętnością wygenerowania wyjątku

Sytuacje wyjątkowe w Pythonie

Obsługa wyjątków polega na wykonaniu kodu pod rygorem klauzuli `try`

```
try:
    ...
```

oraz obsługi sytuacji wyjątkowej w ramach klauzuli `except:` oraz `finally:`

```
except:
    ... # kod który wykonuje się gdy wyjątek wystąpi
else:
    ... # kod który wykonuje się gdy wyjątek nie wystąpi
finally:
    ... # kod który wykonuje się zawsze
```

Przykład obsługi błędów w Pythonie

```
d1 = {0:"zero", 1:"jeden", 2:"dwa", 3:"trzy"}

for key in d1.keys():
    print key

try:
    print d1[4] # dostep do nieistniejacego klucza: KeyError
except:
    print "Nie ma takiego klucza w tym slowniku"
```

Problems | Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python4/nesting/src/p4_except.py
0
1
2
3
Nie ma takiego klucza w tym slowniku
```

Informacja o sytuacji wyjątkowej

Wyjątek to dowolny obiekt(w szczególności zdefiniowany przez użytkownika)

Może nieść ze sobą informację o tym co zaszło, dlatego chcemy mieć do niego dostęp

```
try:
    ...
    ...
    ...
except KeyError as err:
    print type(err)
    print err.args
```

Hierarchia wyjątków

Wyjątki tworzą hierarchię
Pozwala to obsługiwać klasy problemów w jednej klauzuli except

```
try:
    ...
except (KeyError, IndexError) as err:
   pass
except IOError as err:
    print ”Problem dostepu do pliku”
except Exception as err:
    print ”Nieznany wyjatek”
finally:
    f.close()
```

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

Zasady

- Klauzula `try` tworzy „testbed” dla kodu.
- Klauzula `except: ... else: ...` wykona się gdy kod wygeneruje wyjątek.
- Klauzula `finally` wykona się gdy **zawsze.**

Zarządzanie zasobami w obecności wyjątków

- (Plik) jeśli open podniesie `IOError` to
  - `close` się nie wykona i
  - plik nigdy nie zostanie zwolniony:

```
f = open(”/etc/passwd”)
f.close()
```

- kod powinien wyglądać co najmniej tak

```
f = None
try:
    f = open(”/etc/passwd”)
finally:
    if f: f.close()
```

Zarządzanie zasobami w obecności wyjątków

- W praktyce zazwyczaj chcemy wiedzieć:

```
f = None
try:
    f = open(”/etc/passwd”)
except IOError as err:
    print err:
finally:
    if f: f.close()
```

- Klauzula with daje rozwiązanie eleganckie

```
try:
    with open(”/etc/passwd”) as f:
        pass
except IOError as err:
    print err:
```

PRZYKŁAD

P

```
users = {}
with open("/etc/passwd") as passwd_file:
    for line in passwd_file:
        import string
        fields = string.split(line, ":")
        try:
            users[fields[0]] = {"uid":fields[2],
                                "gid":fields[3],
                                "home":fields[5],
                                "shell":fields[6]}
        # using exception to skip lines that did not parse
        except IndexError as err:
            continue

try:
    print "Uzytkownik root ma uid:", users["root"]["uid"], "i gid:", users["root"]["gid"]
    print users["kberezow"]["uid"]
except KeyError as err:
    print "Uzytkownik", err.message, "nie istnieje w systemie"
except Exception:
    print "Nieznany blad"
except:
    print "Nieznana sytuacja wyjatkowa"
else:
    print "Program nie wygenerowal zadnych sytuacji wyjatkowych"
```

Problems Console

```
<terminated> /Users/kberezow/Documents/Python WCSS/python4/nesting/src/p5_except2.py
Uzytkownik root ma uid: 0 i gid: 0
Uzytkownik kberezow nie istnieje w systemie
```

Zgłaszanie wyjątków

- słowo kluczowe `raise`

```
try:
   raise ”Wyjatkowy napis!”
except:
   print ”Wyjatek zlapany!”
```

- hierarchia Exception - wyjątki wbudowane

```
try:
   raise Exception(”Wyjatkowy napis!”)
except Exception as err:
   print err
```

Definiowanie wyjątków

- definiowanie wyjątków poprzez typy własne

```
class E:
    pass
try:
    raise E()
except E as e:
    print e
```

- Lepiej, poprzez rozszerzanie standardowej hierarchii

```
class PotwornyBlad(Exception):
    def __init__(self,value):
        self.value = value
    def __str__(self):
        return repr(self.value)

try:
    raise PotwornyBlad("Nie wiadomo co zrobic")
except Exception as err:
    print err
```

PRZYKŁAD