Programavimo kalba

Python

ketvirtoji paskaita

Marius Gedminas
<mgedmin@b4net.lt>

http://mg.b4net.lt/python/

Matematikos ir Informatikos fakultetas

2006 ruduo

Išskirtinės situacijos

f = file('neratokio')

```
z = x / y # y == 0
```

obj.eat()

```
int('not a number')
```

Exception klasė

Klaidų sukūrimas

raise Exception('man skauda pilvą')

Klaidų sugavimas

```
try:
    f = file('nonono')
except IOError:
    print 'ai ai ai'
```

```
try:
    f = file('nonono')
except IOError, e:
    print 'ai ai ai:', e
```

```
try:
    f = file('nonono')
except IOError, e:
    print 'ai ai ai:', e
else:
    f.read()
```

Resursų atlaisvinimas

(nesvarbu, buvo klaida, ar nebuvo)

```
try:
    ...
finally:
    f.close()
```

Klaidos užbaiginėja funkcijas, kol nebus sugautas

arba užbaigia programą

Kai kurios standartinės klaidos

IOError

įvedimo/išvedimo klaida (nėra failo, nepavyko sukurti katalogo ir pan.)

OSError

operacinės sistemos klaida

IndexError

sąrašo indeksas užeina už ribų

KeyError

žodyne nėra tokio rakto

TypeError

netinkamas argumento tipas

ValueError

netinkama argumento reikšmė

AttributeError

objektas neturi tokio atributo

NameError

nėra tokio kintamojo

SyntaxError

sintaksės klaida Python programoje

ImportError

nepavyko importuoti modulio

ZeroDivisionError

dalyba iš nulio

RuntimeError

programos veikimo klaida (pvz., amžina rekursija)

KeyboardInterrupt

vartotojas paspaudė Ctrl+C

```
try:
    f = file('nonono')
except (OSError, IOError), e:
    print 'ai ai ai:', e
```

```
try:
    f = file('nonono')
except (OSError, IOError):
    print 'ai ai ai'
```

```
try:
    ...
except:    # blogas pavyzys
    f.close()
```

Gaudyti reikia tik tas klaidas, kurių priežastis aiški

Kitaip galima nepastebėti klaidos programoje

arba netyčia neleisti vartotojui su Ctrl+C nutraukti programos

Savo sukurtos klaidos

```
class ManoKlaida(Exception):
    """Negerai tas ir tas"""
```

Apie klaidas viskas.

Programos vykdymo modelis

Sakiniai vykdomi paeiliui

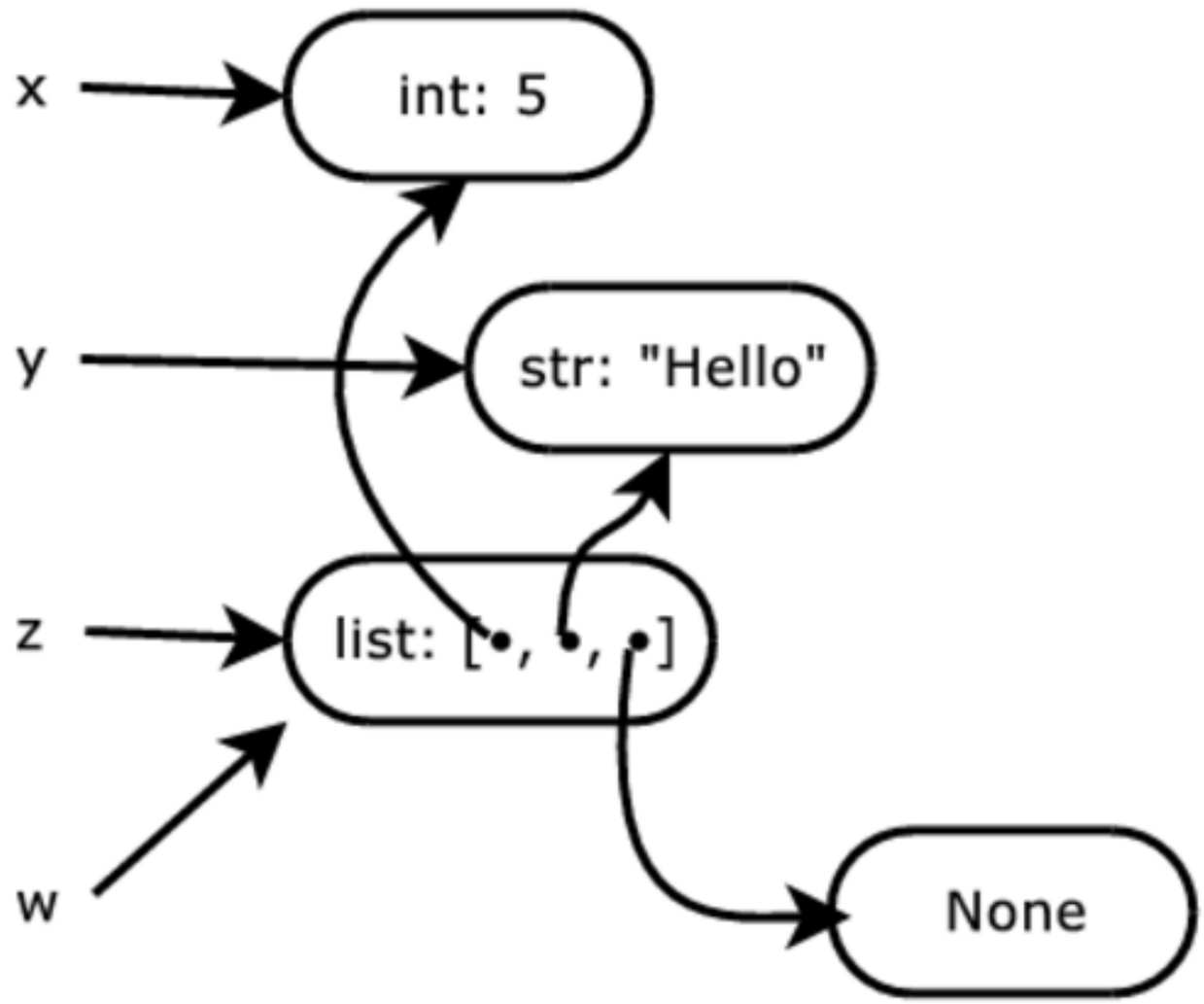
import, class, def

tiesiog sakiniai

Cikliniai importai

kartais veikia, bet dažniausiai yra skausmingi

Vardų erdvės


x
y
z
w
int: 5
str: "Hello"
list: [•, •, •]
None

(prisimenat skirtumą tarp vardų ir objektų?)

Kiekvienas modulis turi savo

Funkcijos ir lokalūs kintamieji

Yra globalių kintamųjų erdvė,
yra lokalių kintamųjų erdvė

Funkcijos funkcijų viduje

```
def katalogu_dydziai(saknis):
    pradzia = len(saknis)
    def spausdink(kur):
        kiek = len(os.listdir(kur))
        kur = kur[pradzia:]
        print "%s: %s" % (kur, kiek)
    apeik_katalogu_medi(saknis, spausdink)
```

Uždariniai
(closure)

```
def pagamink_sifra(raktas):
    def sifruok(tekstas):
        ...
    return sifruok
sifras1 = pagamink_sifra(1)
sifras2 = pagamink_sifra(2)
```

Globalūs kintamieji funkcijose

Funkcijos viduje galima pasiekti tiek globalius, tiek lokalius

Funkcijos viduje galima pakeisti tik lokalius

Nebent naudoji sakinį global

Nutylėtieji funkcijų parametrai

```
def fib(n, cache={}):
    if n <= 2:
        return 1
    if n not in cache:
        cache[n] = fib(n-1, n-2)
    return cache[n]
```

Su jais atsargiai!

```
class Zaidejas(object):
    def __init__(self, maisas=[]):
        self.maisas = [] # klaida!?

jonas = Zaidejas()
petras = Zaidejas()
jonas.maisas is petras.maisas # !
```

```
class Zaidejas(object):
    def __init__(self, maisas=None):
        if maisas is None:
            maisas = []
        self.maisas = maisas
```

arba

```
class Zaidejas(object):
    def __init__(self, maisas=[]):
        self.maisas = list(maisas)
```

Apibendrinimas

Sakiniai vykdomi paeiliui

Moduliai turi savo vardų erdves

Funkcijų vardų erdvės įsideda viena į kitą