Programavimo kalba

Python

septintoji paskaita

Marius Gedminas
<mgedmin@b4net.lt>

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

Matematikos ir Informatikos fakultetas

2006 ruduo

Antys, tipai, protokolai, iteratoriai, generatoriai

Statinis tipizavimas,

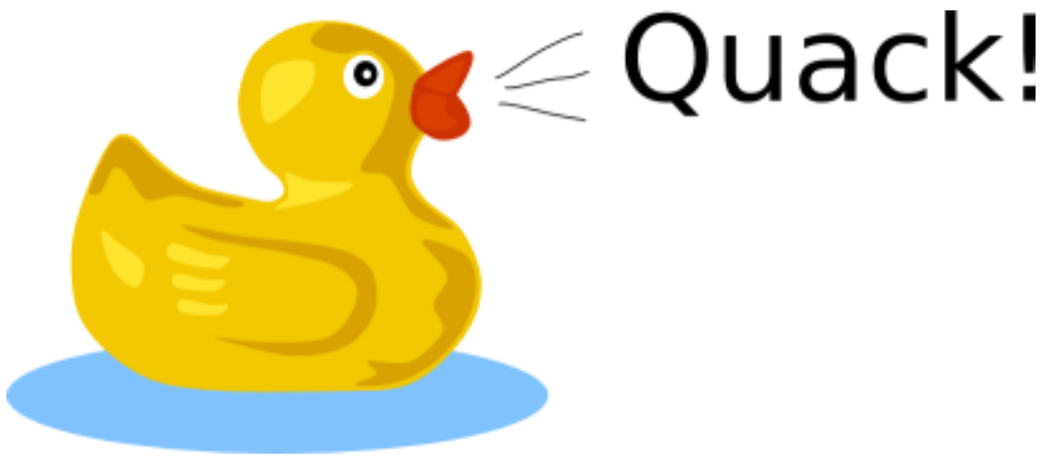
Dinaminis tipizavimas,

Griežtas tipizavimas,

Silpnas tipizavimas

Antiškas tipizavimas

(Duck typing)



If it looks like a duck and quacks like a duck, it must be a duck.

Jei objektas turi reikiamus atributus/metodus, reiškia jis mums tinka

Geriau naudoti hasattr/gaudyti exceptionus nei tikrinti type() ar isinstance()

Objektai, panašūs į file:

StringIO

GzipFile

Soketai

...

Jie nėra 100% atitikmenys

(pvz., socketai neturi tell metodo)

Privalumas: gali realizuoti tik dalį metodų ir turėti naudingą klasę

Trūkumas: kompiliatorius neperspės, jei užmirši realizuoti dalį metodų

Nieko naujo: vienos kalbos stengiasi

apsaugoti neatidžius programuotojus

tramdomaisiais marškiniais,

kitos stengiasi netrukdyti dirbti.

Objektai, panašūs į file...

kaip pavadinti jų panašumus?

Protokolu

file protokolas:

read()
write()
flush()
close()
readline()
readlines()
writelines()
seek()
tell()
truncate()

Kokie dar yra protokolai?

Sekos protokolas
(vartotojo pusė)

x in s
x not in s
len(s)
s[i]
s[i:j]
s[i:j:k]
s * n, n * s
s + t
for x in s: ...

Sekos protokolas
(rašytojo pusė)

__contains__
__len__
__getitem__
__getslice__
__mul__, __rmul__
__add__, __radd__

Atvaizdžio protokolas
(vartotojo pusė)

x in s
x not in s
len(s)
s[k]
s.keys()
s.values()
s.items()
for x in s: ...

Atvaizdžio protokolas
(rašytojo pusė)

__contains__
__len__
__getitem__
keys
values
items
__iter__

UserDict, UserList moduliai padeda, jei norite rašyti savo sekas ar atvaizdžius

Iteravimo protokolas
(vartotojo pusė)

for x in s: ...

iter(s)

Iteravimo protokolas
(rašytojo pusė)

__iter__

Iteratoriai

Yra seka, per kurią galima prabėgti.

Iteratorius atsimena vietą toje sekoje

Analogai: rodyklės, masyvo indeksai

```
for x in s:
    print x
```

```
# Jei s yra seka
i = 0
while True:
    try:
        x = s[i]
        i += 1
    except IndexError:
        break
print x
```

```
# Jei s palaiko iteratoriaus
# protokolą
i = iter(s)
while True:
    try:
        x = i.next()
    except StopIteration:
        break
    print x
```

Iteratorius:

i.next()

duok kitą elementą arba mesk StopIteration

iter(i)

grąžink **i** (kad for ciklui būtų galima paduoti iteratorių)

Realizacija

```
class MyIter:
    def __iter__(self):
        return self

    def next(self):
        ...
```

Realizacija / naudojimas

```
def __iter__(self):                    iter(i)
def __len__(self):                     len(i)
def __mul__(self, j):                  i * j
```

Nedarykite!

```
x = s.__len__()
s = w.__str__()
c = a.__add__(b)
```

Darykite

```
x = len(s)
s = str(w)
c = a + b
```

Iteratoriai

Generatoriai

Pigus būdas rašyti iteratoriams

Generatoriaus pavyzdys

```
def oddsquares(s):
    for x in s:
        if x % 2 == 1:
            yield x ** 2
```

Generatoriaus naudojimas

```
for n in oddsquares(range(10)):
    print n
```

Generatoriaus privalumai

Nekuria ilgo sąrašo atmintyje

Iškviečiamas tik tiek kartų, kiek reikia

Galima „išsukti“ algoritmą išvirkščiai

os.walk yra generatorius,

„išverčiantis“ katalogų

medžio apėjimą išvirkščiai

Kitas pavyzdys

```
def fibonacci():
    a = b = 1
    while True:
        yield a
        a, b = b, a+b
```

Begalinis generatorius!

```
for n in fibonacci():
    print n
    if n > 1000:
        break
```

enumerate

```
f = file('/etc/motd')
for n, s in enumerate(f):
    print "%5d: %s" % (n, s.rstrip())
```

Jei enumerate nebūtų

```
def enumerate(s):
    n = 0
    for x in s:
        yield (n, x)
        n += 1
```

itertools modulis

itertools modulis

count([n]) -> n, n+1, n+2 ...
cycle(p) -> p0, p1, ... plast, p0, ...
repeat(elem[, n]) -> elem, elem, elem ...
tee(i) -> i1, i2
chain(p, q) -> p0, p1, ... plast, q0, ...
izip(p, q) -> (p0, q0), (p1, q1) ...
ifilter(pred, s) -> filtravimas
imap(fun, p) -> fun(p0), fun(p1) ...
...

Negražus triukas

```
def pairs(s):
    i = iter(s)
    return itertools.izip(i, i)
```

```
>>> list(pairs([1, 2, 3, 4, 5, 6, 7]))
[(1, 2), (3, 4), (5, 6)]
```

Rekursyvūs generatoriai

Medžio apėjimas (printinam viską)

```
def preorder_tree_walk(node):
    print node
    for child in node.children:
        preorder_tree_walk(child):
```

Medžio apėjimas (generatorius)

```
def preorder_tree_walk(node):
    yield node
    for child in node.children:
        for node in preorder_tree_walk(
                                    child):
            yield node
```

Iki