Python klasės ir klaidų apdorojimas

Albertas Gimbutas

[1]Matematikos ir informatikos institutas
Vilniaus universitetas

2014 pavasaris

Turinys

Datos tipas

```
>>> from datetime import datetime
>>> now = datetime.now()
datetime.datetime(2014, 3, 17, 8, 37, 29, 556397)
>>> print(now)
2014-03-17 08:37:29.556397
>>> start.strftime('%Y-%m-%d')
'2014-03-17'
>>> datetime.strptime('2014-03-17', '%Y-%m-%d')
datetime.datetime(2014, 3, 17, 0, 0)
```

- python.org: datos tipas
- python.org: datos formatavimas

Trukmės tipas

```
>>> from datetime import datetime, timedelta
>>> start = datetime.now()
datetime.datetime(2014, 3, 17, 8, 37, 29, 556397)
>>> datetime.now() - start
datetime.timedelta(0, 152, 71658)
>>> print(delta * 1000) # + - * / // %
1 day, 18:14:31.658000  # [D day[s], ][H]H:MM:SS[.UUUUUU]
>>> delta.total_seconds()
45621.4974
>>> timedelta(seconds=45621.4974)
1 day, 18:14:31.658000
```

- python.org: trukmės tipas (timedelta)

Generatoriai

- Generatorius yra funkcija grąžinanti objektą per kurį galima iteruoti (iteratorių).

```
>>> def mano_generatorius(n=1):
...     for i in range(n):
..         yield i**2
>>> g = mano_generatorius()
>>> g.next()
0
>>> g.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>> [e for e in mano_generatorius(4)]
[0, 1, 4, 9]
```

Klasės apibrėžimas

```
>>> class A:
...     count = 10
...     def __init__(self, count=None, *args, **kwargs):
...         if count:
...             self.count = count
...     def increase_count(self, x=5):
...         self.count += x
...         return self.count
...
>>> a = A()
>>> b = A(count=20)
>>> a.increase_count()
15
>>> a.count + b.count
35
```

- python.org: klasės

Daugybinis paveldėjimas

```
>>> class A:
...     x = 1
...     def __init__(self, x):
...         self.x = x
```

```
>>> class B:
...     y = 2
...     def __init__(self, y, *args, **kwargs):
...         self.y = y
...         super(B, self).__init__(y, *args, **kwargs)
```

```
>>> class C(A, B):
...     pass
>>> c = C(3)      # c.x == 3, c.y == 2
>>> class D(B, A):
...     pass
>>> d = D(3)      # d.x == 3, d.y == 3
```

Klasės funkcionalumo praplėtimas (Mixin)

```
>>> class PositiveMixin:
...     def __init__(self, v, *args, **kwargs):
...         if v < 0:
...             raise ValueError('Reikšmė neigiama')
...         super(PositiveMixin, self).__init__(v, *args,
...                                             **kwargs)
```

```
>>> class E(PositiveMixin, B, A):
...     pass
```

```
>>> e = E(-1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 4, in __init__
ValueError: Reikšmė neigiama
>>> e = E(3)    # e.x == 3, e.y == 3
```

Klasės kintamieji

```
>>> class A:
...     x = 1
...     @classmethod
...     def get_class_x(cls):
...         return cls.x
...     def get_obj_x(self):
...         return self.x
...
>>> a = A()    # A.x == 1    a.x == 1
>>> A.x = 2    # A.x == 2    a.x == 1
>>> a.x = 3    # A.x == 2    a.x == 3
>>> A.get_class_x(), a.get_obj_x()
(2, 3)
>>> a = A()    # A.x == 2    a.x == 2
```

- Privatūs kintamieji ir metodai Python kalboje neegzistuoja. Vidinių (privačių) metodų ir kintamųjų pavadinimai turėtų prasidėti simboliu _, pavyzdžiui `_foo`.

Klaidų sukėlimas

```
>>> 1 / 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
```

```
>>> raise ValueError('Kilusios klaidos pranešimas')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: Kilusios klaidos pranešimas
```

- python.org: dažniausiai kylančių klaidų sąrašas

Klaidų apdorojimas

```
>>> try:
...     f = open(filename, 'r')
... except IOError:    # Visada nurodyti klaidos tipą
...     print('Nepavyko atidaryti failo')
... except (ValueError, RuntimeError, Exception) as e:
...     print('Kilo klaida:', e)
... except:
...     print('Įvyko nenumatyta klaida')
...     raise      # Sukelia tą pačią klaidą
... else:     # Blokas vykdomas jeigu nekilo klaidų
...     print f.readline()
... finally: # Blokas visada įvykdomas
...     f.close()
```

- python.org: klaidų apdorojimas

Klaidų klasių kūrimas

```
>>> class SimpleError(Exception):
...     def __init__(self, value):
...         self.value = value
...     def __str__(self):
...         # Grąžina tekstinę objekto reprezentaciją
...         return repr(self.value)
```

```
>>> class ExtendedError(MySimpleError):
...     def __init__(self, value, *args):
...         super(MyExtendedError, self).__init__(value)
...         self.args = args
```

```
>>> try:
...     raise ExtendedError('Ne tinkamas kontekstas',
...                         context, a, b)
... except ExtendedError as e:
...     context, a, b = e.args    # Sutvarkomas kontekstas
```

Moduliai

- Modulio direktorijoje turi egzistuoti `__init__.py` failas, kuriame rašoma modulio inicializavimo logika.

```
from project.my_module.my_file import MyClass
from .my_file import MyClass
```

Paketo registracija PyPI indekse:

1. Prisiregistruoti: https://pypi.python.org/pypi?%3Aaction=register_form
2. Sukurti modulio meta-duomenų failus: setup.py, README.rst ir kt.
3. Įvykdyti komandą: `python setup.py register upload`

- M. Gedminas "Python paketai"
- python.org: paketų indeksas PyPi
- Python paketų įkėlimas į pypi.python.org