Akademia Pythona

VIII Zagadnienia zaawansowane

KN Pythona - Kurs Pythona

KN Pythona wita na kursie Pythona.

Plan:

- Zarządzane atrybuty

Po co zarządza się atrybutami?

```
person.name # pobranie wartości
person.name = wartość # przypisanie wartości
```

Czym jest obiekt?

```
{
    atrybut
    metoda
    {
        atrybut
    }
}
```

Zarządzanie dostępem

```
class Person:
    def get_name(self):
        if not valid(self.name):
            raise TypeError()
        else:
            return self.name.transform()
    def set_name(self, value):
        if not valid(value):
            raise TypeError()
        else:
            self.name = transform(value)
```

Kod wykonywalny w miejscu dostępu

Inne podejścia:

- Metody *getattr* oraz *setattr*
- Metoda *getattribute*
- Funkcja property
- Protokół deskryptora

Właściwości

```
attribute = property(fget, fset, fdel, doc)
```

Właściwości

```
class Person:
    def __init__(self, name):
        self._name = name
    def get_name(self):
        return self._name
    def set_name(self, value):
        self._name = value
    def del_name(self):
        del self._name
    name = property(get_name, set_name, del_name, 'name')
```

Obliczanie atrybutów

```
class PropSquare:
    def __init__(self, start):
        self.value = start
    def get_x(self):
        return self.value ** 2
    def set_x(self, value):
        self.value = value
    x = property(get_x, set_x)
```

Właściwości jako dekoratory

```
class Person:
    def __init__(self, name):
        self._name = name
    @property
    def name(self):
        "Dokumentacja name"
        return self._name
    @name.setter
    def name(self, value):
        self._name = value
    @name.deleter
    def name(self):
        del self._name
```

Deskryptory

Właściwości są uproszczonymi deskryptorami.

Deskryptory

```
class Descriptor:
    'Doc string'
    def __get__(self, instance, owner): ...
    def __set__(self, instance, value): ...
    def __delete__(self, instance): ...
```

Deskryptory

```
class Descriptor:
    def __get__(self, instance, owner):
        print(self, instance, owner)
class Subject:
    attr = Descriptor()
x = Subject()
x.attr
# <..main..Descriptor <main subject <class Subject
Subject.attr
# <..main..Descriptor None <class Subject
```

Deskryptory

```
class D:
    def __get__(*args): print('hop')
class C:
    a = D()
X = C()
X.a # hop
C.a # hop
X.a = 99
X.a # 99
C.a # hop
```

Deskryptory tylko do odczytu

```
class D:
    def __get__(*args): return 1234
    def __set__(*args): raise AttributeError()
class C:
    a = D()
x = C()
x.a # 1234
x.a = 12 # AttributeError...
```

Deskrtyptory

```
class Name:
    def __get__(self, instance, owner):
        return instance._name
    def __set__(self, instance, value):
        instance._name = value
    def __delete__(self, instance):
        del instance._name
class Person:
    name = Name()
```

Deskryptory a klasy osadzone

```
class A:
    class B:
        def __get__(*args): return 1234
    a = B()
```

Informacja o stanie w deskryptorze

```
class B:
    def __init__(self):
        self.cnt = 0
    def __get__(*args):
        self.cnt += 1
        print(self.cnt)
        return 1234
class A:
    a = B()
x = A()
x.a # 1 1234
x.a # 2 1234
```

Metody getattr oraz getattribute

Różnica:

- metoda *getattr*: niezdefiniowane atrybuty
- metoda *getattribute*: każdy atrybut

Zarządzanie atrybutami

```
def __getattr__(self, name) # niezdefiniowane
def __getattribute__(self, name) # wszystkie
def __setattr__(self, name, value) # wszystkie
def __delattr__(self, name) # wszystkie
```

Zarządzane atrybuty

```
class Wrapper:
    def __init__(self, object):
        self.wrapped = object
    def __getattr__(self, attrname):
        print('Śledzenie: ' + attrname)
        return getattr(self.wrapped, attrname)
```

Zagadka

```
class A:
    def __getattribute__(self, name):
        return self.other
x = A()
print(x.abc)
```

Zagadka - rozwiązanie

`RecursionError` !

Zagadka - kod

```
class A:
    def __getattribute__(self, name):
        return object.__getattribute__(self, 'other')
```

Pętla nr 2

```
class A:
    def __setattr__(self, attrname, value):
        self.other = value # pętla!
```

Pętla nr 2 - rozwiązanie

```
class A:
    def __setattr__(self, attrname, value):
        self.__dict__['other'] = value
```

Pętla nr 1 - CD

```
class A:
    def __getattribute__(self, attrname):
        return self.__dict__[attrname] # ?
```