Politechnika Świętokrzyska

Laboratorium

Programowanie w języku Python 2

Ćwiczenie 4

Dekoratory
Właściwości

dr inż Robert Kazała

Cel ćwiczenia

Celem ćwiczenia jest poznanie funkcjonalności dekoratorów oraz deklarowania właściwości klas z wykorzystaniem funkcji property().

Dekoratory

Funkcja dekorująca

Przykład 1

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is
called.")
        func()
        print("Something is happening after the function is
called.")
    return wrapper

def say_whee():
    print("Whee!")

say_whee = my_decorator(say_whee)

say_whee()
```

Przykład z wykorzystaniem dekoratora

Przykład 2

```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is
called.")
        func()
        print("Something is happening after the function is
called.")
    return wrapper

@my_decorator
def say_whee():
    print("Whee!")

say_whee()
```

Dekorator z argumentami

Przykład 3

```
def do_twice(func):
    def wrapper_do_twice(*args, **kwargs):
        func(*args, **kwargs)
        func(*args, **kwargs)
    return wrapper_do_twice

@do_twice
def my_print(text):
    print(text)

my_print('test')
```

Biblioteka functools

Dekorator @functools.wraps

Przykład 4

```
from functools import wraps
def my_decorator(f):
    @wraps(f)
    def wrapper(*args, **kwds):
        print('Calling decorated function')
        return f(*args, **kwds)
    return wrapper

@my_decorator
def example():
    """Docstring"""
    print('Called example function')

example()


example.__name__

example.__doc__
```

Przykład użycia @functools.wraps

Przykład 5

```
from functools import wraps
```

```
def my_decorator(func):
     @wraps(func)
     def wrapper(*args, **kwds):
         print('Wywołanie funkcji dekorowanej')
         value = func(*args, **kwds)
         print('Koniec wywołania funkcji dekorowanej')
         return value
     return wrapper

@my_decorator
def example():
    """Dokumentacja funkcji"""
    print('Funkcja dekorowana')

example()
print(example.__name__)
print(example.__doc__)
```

Mierzenie czasu z wykorzystaniem @functools.wraps

Przykład 6

```
import functools
import time

def timer(func):
    """Print the runtime of the decorated function"""
    @functools.wraps(func)
    def wrapper_timer(*args, **kwargs):
        start_time = time.perf_counter()    # 1
        value = func(*args, **kwargs)
        end_time = time.perf_counter()      # 2
        run_time = end_time - start_time    # 3
        print(f"Finished {func.__name__!r} in {run_time:.4f}
secs")
        return value
    return wrapper_timer

@timer
def waste_some_time(num_times):
    for _ in range(num_times):
        sum([i**2 for i in range(10000)])

waste_some_time(5)
```

Funkcja property()

Tworzenie deskryptora z wykorzystaniem funkcji property()

Przykład 7

```
class Person(object):
    def init(self):
        self._name = ''

    def fget(self):
        print "Getting: %s" % self._name
        return self._name

    def fset(self, value):
        print "Setting: %s" % value
        self._name = value.title()

    def fdel(self):
        print "Deleting: %s" %self._name
        del self._name

    name = property(fget, fset, fdel, "I'm the property.")


user = Person()
user.name = 'john smith'

user.name
del user.name
```

Przykład 8

```
# Alphabet class
class Alphabet:
    def __init__(self, value):
        self._value = value

    # getting the values
    def getValue(self):
        print('Getting value')
        return self._value

    # setting the values
    def setValue(self, value):
        print('Setting value to ' + value)
        self._value = value
```

```
    # deleting the values
    def delValue(self):
        print('Deleting value')
        del self._value

    value = property(getValue, setValue, delValue, )

# passing the value
x = Alphabet('GeeksforGeeks')
print(x.value)

x.value = 'GfG'

del x.value
```

Dekorator @property

Tworzenie deskryptora z wykorzystaniem dekoratora @property

Przykład 9

```
class Person(object):

    def init(self):
        self._name = ''

    @property
    def name(self):
        print "Getting: %s" % self._name
        return self._name

    @name.setter
    def name(self, value):
        print "Setting: %s" % value
        self._name = value.title()

    @name.deleter
    def name(self):
        print ">Deleting: %s" % self._name
        del self._name


user = Person()
user.name = 'john smith'

user.name
del user.name
```

Przykład 10

```
class Alphabet:
    def __init__(self, value):
        self._value = value

    # getting the values
    @property
    def value(self):
        print('Getting value')
        return self._value

    # setting the values
    @value.setter
    def value(self, value):
        print('Setting value to ' + value)
        self._value = value

    # deleting the values
    @value.deleter
    def value(self):
        print('Deleting value')
        del self._value


# passing the value
x = Alphabet('Peter')
print(x.value)

x.value = 'Diesel'

del x.value
```

Literatura

https://docs.python.org/3/reference/datamodel.html#special-method-names

Zadania

1. Uruchomić, przeanalizować, omówić efekt działania i zmodyfikować wszystkie przykłady z wykładu.
2. Porównać działanie poniższego kodu i kodu z przykładu 4. Omówić efekt wykorzystania dekoratora @wraps.

```
from functools import wraps
def my_decorator(f):
    def wrapper(*args, **kwds):
        """Wraper docstring"""
        print('Calling decorated function')
        return f(*args, **kwds)
    return wrapper

@my_decorator
def example():
    """Docstring"""
    print('Called example function')

example()


example.__name__

example.__doc__
```

3. Utworzyć własne przykłady wykorzystujące dekoratory.
4. Utworzyć własne przykłady wykorzystujące funkcję property().
5. Utworzyć własne przykłady wykorzystujące dekorator property().