Katando Python III

isra@miscorreos.org
@kamaxeon

Agenda

- **Decorator**
- **Classes**

Decorators

```
def my_decorator(func):
    def wrapper():
        print("Before")
        func()
        print("After")
    return wrapper


def say_hello():
    print("Hello Katando Python!")


say_hello = my_decorator(say_hello)
say_hello()
>>> Before
Hello Katando Python !
After
```

Decorators

```
def my_decorator(func):
    def wrapper():
        print("Before")
        func()
        print("After")
    return wrapper

@my_decorator
def say_hello():
    print("Hello Katando Python!")

say_hello()
>>> Before
Hello Katando Python !
After
```

Decorators

```
@app.route('/secret_page')
@login_required
def secret_page():
    ....
```

```
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if g.user is None:
            return redirect(url_for('login', next=request.url))
        return f(*args, **kwargs)
    return decorated_function
```

Decorators - classes

```
class logger(object):

    def __init__(self, fn):
        print("Logger is instantiated in the definition of the function")
        self.fn = fn

    def __call__(self, *args):
        print("Function name {}".format(self.fn.__name__))
        for i, arg in enumerate(args):
            print("arg %d:%d" % (i, arg))
        return self.fn(*args)


@logger
def adder(*args):
    return sum([i for i in args])


adder(1, 2, 3, 4)
```

```
>>>Logger is instantiated in the definition of the function
Function name adder
arg 0:1
arg 1:2
arg 2:3
arg 3:4
```

classes

```
class Example(object):

    def __init__(self, x):
        self.x = x


example = Example(4)
print(example.x)

>>> 4
```

classes inheritance

```
class Person(object):

    def __init__(self, first, last):
        self.firstname = first
        self.lastname = last


class Employee(Person):

    def __init__(self, first, last, staffnum):
        Person.__init__(self, first, last)
        self.staffnumber = staffnum

    def getEmployee(self):
        return "{} {} {}".format(self.firstname, self.lastname,
                                 self.staffnumber)


james = Employee('James', 'Bond', '007')
print(james.getEmployee())

>>> James Bond 007
```

classes encapsulation

```
class Example(object):

    def __init__(self, a, b, c):
        self.a = a
        self._b = b
        self.__c = c

    def print_values(self):
        print('x = {}, _x = {}, __x = {}'.format(self.

obj = Example(2, 3, 4)
obj.print_values()
>>> a = 2, _b = 3, __c = 4
obj.a = 5
obj._b = 6
obj._Example__c = 7
obj.print_values()
>>> a = 5, _b = 6, __c = 7
```

classes polymorphism

```
class A():

    def __init__(self):
        self.__x = 1

    def m1(self):
        print("m1 inside of  A")


class B(A):

    def __init__(self):
        self.__y = 1

    def m1(self):
        print("m1 inside of B")


c = B()
c.m1()

>>> me inside of B
```

classes methods

```
class MyClass:
    def method(self):
        return 'instance method called', self

    @classmethod
    def classmethod(cls):
        return 'class method called', cls

    @staticmethod
    def staticmethod():
        return 'static method called'
```

classes classmethods

```
class Pizza(object):
    def __init__(self, ingredients):
        self.ingredients = ingredients

    @classmethod
    def margherita(cls):
        return cls(['mozzarella', 'tomatoes'])

    @classmethod
    def prosciutto(cls):
        return cls(['mozzarella', 'tomatoes', 'ham'])
```

classes staticmethods

```
import math

class Pizza:
    def __init__(self, radius, ingredients):
        self.radius = radius
        self.ingredients = ingredients

    @staticmethod
    def circle_area(r):
        return r ** 2 * math.pi

>>> Pizza.circle_area(4)
50.2648…..
```

classes @property

```
class Example(object):

    def __init__(self, x):
        self.x = x

    @property
    def x(self):
        return self.__x

    @x.setter
    def x(self, x):
        if x < 100:
            self.__x = x


example = Example(10)
print(example.x)
>>>10
example.x = 90
print(example.x)
>>> 90
example.x = 900
print(example.x)
>>> 90
```

classes magic methods

Construction:

- __init__
- __del__

Comparison:

- __cmp__
- __eq__
- __ne__
- __lt__
- __gt__
- __le__
- __ge__

Arithmetic:

- __add__
- __sub__
- __mul__
- __floordiv__
- __div__
- __mod__
- __divmod__
- __pow__
- __lshift__
- __rshift__
- __and__
- __or__
- __xor__

Augmented assignment:

- __iadd__
- __isub__
- __imul__
- __ifloordiv__
- __idiv__
- __imod__
- __ipow__
- __ilshift__
- __irshift__
- __iand__
- __ior__
- __ixor__

Behind containers:

- __len__
- __getitem__
- __setitem__
- __delitem__
- __iter__
- __reversed__
- __contains__
- __missing__

Repesenting:

- __str__
- __repr__
- __hash__

Links: https://rszalski.github.io/magicmethods/

Questions ?

References

- https://www.datacamp.com/community/tutorials/python-list-comprehension
- https://snakify.org/en/lessons/two_dimensional_lists_arrays/
- https://www.digitalocean.com/community/tutorials/how-to-use-args-and-kwargs-in-python-3
- https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html