onnet

2021
VIE

Odoo Method Decorators

www.on.net.vn
www.arrowhitech.com

onnet
BY AHT TECH JSC

Table of contents

- Higher order function
- First class object
- Python decorator
- Odoo ORM API
- Practice

onnet
BY AHT TECH JSC

Python decorator: Higher order function

In mathematics and computer science, a **higher-order function** is a **function** that does at least one of the following:

- Takes one or more functions as arguments
- Returns a function as its result.

Higher order function

Properties of higher-order functions:

- A **function** is an instance of the **Object type**.
- You can store the function in a **variable**.
- You can pass the function as a **parameter** to another function.
- You can **return the function** from a function.
- You can store them in **data structures** such as hash tables, lists, ...

Higher order function

Example:

```
def shout(text):
    return text.upper()

def whisper(text):
    return text.lower()

def greet(func):
    # storing the function in a variable
    greeting = func("Hi, I am created by a function passed as an argument.")
    print(greeting)

greet(shout)
greet(whisper)
```

First class object

A **first-class object** is an **entity** within a programming language that can:

- Appear in an **expression**
- Be assigned to a **variable**
- Be used as an **argument**
- Be returned by a **function call**

First class object

A programming language is said to support **first-class functions** if it treats functions as **first-class objects**. Python supports the concept of **First Class functions**.

First class object

Example:

```
def shout(text):
     return text.upper()
     print(shout('Hello'))
yell = shout

print (yell('Hello'))
```

Python decorator

A **decorator** is a **function** that takes **another function** and extends the behavior of the later function without explicitly **modifying** it.

Python decorator

Example:

```
@foo
def bar():
    print("barz")

Above code is equivalent to:

def bar():
    print("barz")

bar = foo(bar)
```

Python decorator

odoo Gold Partner

Example:

```
import time
import math

def calculate_time(func):
    def inner1(*args, **kwargs):
        begin = time.time()
        func(*args, **kwargs)
        end = time.time()
        print("Total time taken in : ", func.__name__, end - begin)
    return inner1

@calculate_time
def factorial(num):
    print(math.factorial(num))
factorial(10)
```

onnet BY AHT TECH JSC

Odoo decorator: Depends

@api.depends

- Return a decorator that specifies the field dependencies of a "compute" method.
- The function defined with this decorator will be called if any change happens in the fields specified.
- The change to the field can be from ORM or changes in the form.
- If a compute function value depends on another field, then it must be specified using depends. In addition, the depends attribute can also be dotted field names such as ‘product_id.categ_id’.

Example:

```
pname = fields.Char(compute='_compute_pname')

@api.depends('partner_id.name', 'partner_id.is_company')
def _compute_pname(self):
      for record in self:
            if record.partner_id.is_company:
                  record.pname = (record.partner_id.name or "").upper()
            else:
                  record.pname = record.partner_id.name
```

Odoo decorator: Onchange

@api.onchange

- The function of this decorator will be called when the field value changes.
- Supports only single field names, dotted names will not be considered.
- Can be invoked on pseudo-records that contain values of the form.
- Can return a notification.

Example:

```
    @api.onchange('partner_id')
    def _onchange_partner(self):
        self.message = "Dear %s" % (self.partner_id.name or "")
        return {
            'warning': {
                'title': "Warning",
                'message': "What is this?",
                'type': 'notification'
            }
        }
```

Odoo decorator: Model create multi

Gold Partner

@api.model_create_multi

The function defined with this decorator takes a list of dictionaries and creates multiple records. Moreover, the method can be called with a single or list of dictionaries

```
@api.model_create_multi
def create(self, vals_list):
    records = super(Foo, self).create(vals_list)
    …
    return records
```

onnet
BY AHT TECH JSC

Odoo decorator: Model

@api.model

Decorate a record-style method where **`self`** is a recordset, but its contents is not relevant, only the model is.

Example:

```
@api.model
def _get_default_date(self):
     return fields.Date.today()
```

Odoo decorator: Constrains

@api.constrains

Decorate a constraint checker.

Each argument must be a field name used in the check:

```
@api.constrains('name', 'description')
def _check_description(self):
    for record in self:
        if record.name == record.description:
            raise ValidationError("Fields name and description must be different")
```

Invoked on the records on which one of the named fields has been modified.

Practice

Exercise 1:

Create model **x.student** with 2 fields **first_name** and **last_name**. Let's add a new field **full_name** which will be auto displayed when key in both **first_name** and **last_name** by combining them.
*For example: If **first_name** is Peter and **last_name** is Parker then the **full_name** will be Peter Parker*

Exercise 2:

Add a new field **date_of_birth** to **x.student** and make sure student's age must be in range 14 to 18.
Hint: use python module "datetime" to calculate student's age

Q&A