onnet

2021
VIE

Odoo Coding Guidelines

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

onnet
BY AHT TECH JSC

Table of contents

01

Module structure

Directories

```
crm
|-- data: demo and data xml
|-- models: models def
|-- controllers: HTTP routes
|-- views: views and templates
|-- static: web assets
|-- security: access rights and record rules
|-- report
|-- security
|-- tests
|-- wizard
|-- i18n: translations
|-- __init__.py
|-- __manifest__.py
```

01

Module structure

File naming - models

```
models
|-- crm_lead.py
|-- crm_lost_reason.py
|-- crm_stage.py
|-- crm_team.py
|-- res_partner.py
|-- res_users.py
...
```

Split the business logic by sets of models belonging to a same main model

01

Module structure

File naming - security

```
security
|-- crm_security.xml
|-- ir.model.access.csv
```

01

Module structure

File naming - views

views

```
|-- assets.xml
|-- crm_lead_views.xml
|-- crm_lost_reason_views.xml
|-- crm_menu_views.xml
|-- crm_stage_views.xml
|-- crm_team_views.xml
|-- res_partner_views.xml
```

- backend views: <model>_views.xml
- menus: <module>_menus.xml
- templates: <model>_template.xml
- bundles: assets.xml

01

Module structure

File naming - views

```
views
|-- assets.xml
|-- crm_lead_views.xml
|-- crm_lost_reason_views.xml
|-- crm_menu_views.xml
|-- crm_stage_views.xml
|-- crm_team_views.xml
|-- res_partner_views.xml
```

- backend views: <model>_views.xml
- menus: <module>_menus.xml
- templates: <model>_template.xml
- bundles: assets.xml

01

Module structure

Actually in Odoo 15 (and after)

views

```
|-- ~~assets.xml~~
|-- crm_lead_views.xml
|-- crm_lost_reason_views.xml
|-- crm_menu_views.xml
|-- crm_stage_views.xml
|-- crm_team_views.xml
|-- res_partner_views.xml
```

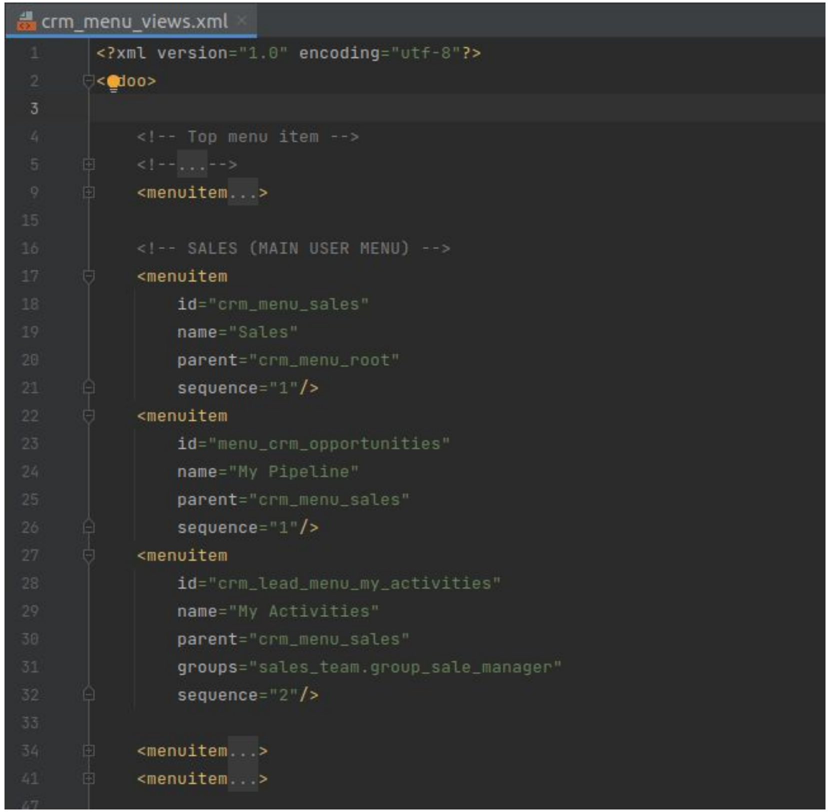
__manifest__.py

```
'assets': {
   'web.assets_qweb': [
       'crm/static/src/xml/forecast_kanban.xml',
   ],
   'web.assets_backend': [
       'crm/static/src/js/crm_form.js',
       'crm/static/src/js/crm_kanban.js',
       'crm/static/src/scss/crm.scss',
       ...
   ],
   'web.assets_tests': [
       'crm/static/tests/tours/**/*',
       ...
   ],
   'web.qunit_suite_tests': [
       'crm/static/tests/mock_server.js',
       ...
   ],
},
```

01

Module structure

crm_menu_views.xml



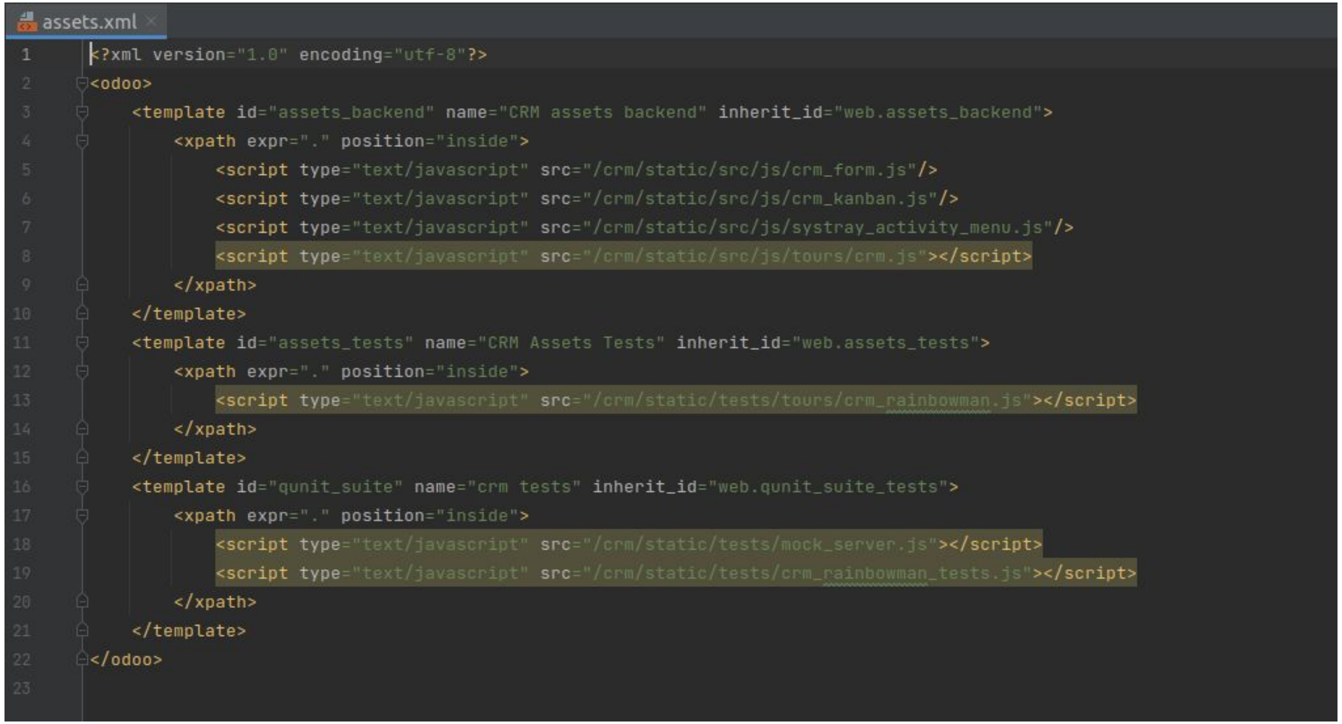
crm_menu_views.xml

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <!-- Top menu item -->
    <!--...-->
    <menuitem...>

    <!-- SALES (MAIN USER MENU) -->
    <menuitem
        id="crm_menu_sales"
        name="Sales"
        parent="crm_menu_root"
        sequence="1"/>
    <menuitem
        id="menu_crm_opportunities"
        name="My Pipeline"
        parent="crm_menu_sales"
        sequence="1"/>
    <menuitem
        id="crm_lead_menu_my_activities"
        name="My Activities"
        parent="crm_menu_sales"
        groups="sales_team.group_sale_manager"
        sequence="2"/>

    <menuitem...>
    <menuitem...>
```

01

Module structure

crm/views/assets.xml (Odoo 14 and before)


assets.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <template id="assets_backend" name="CRM assets backend" inherit_id="web.assets_backend">
        <xpath expr="." position="inside">
            <script type="text/javascript" src="/crm/static/src/js/crm_form.js"/>
            <script type="text/javascript" src="/crm/static/src/js/crm_kanban.js"/>
            <script type="text/javascript" src="/crm/static/src/js/systray_activity_menu.js"/>
            <script type="text/javascript" src="/crm/static/src/js/tours/crm.js"></script>
        </xpath>
    </template>
    <template id="assets_tests" name="CRM Assets Tests" inherit_id="web.assets_tests">
        <xpath expr="." position="inside">
            <script type="text/javascript" src="/crm/static/tests/tours/crm_rainbowman.js"></script>
        </xpath>
    </template>
    <template id="qunit_suite" name="crm tests" inherit_id="web.qunit_suite_tests">
        <xpath expr="." position="inside">
            <script type="text/javascript" src="/crm/static/tests/mock_server.js"></script>
            <script type="text/javascript" src="/crm/static/tests/crm_rainbowman_tests.js"></script>
        </xpath>
    </template>
</odoo>

```

01

Module structure

File naming - data

data

```
|-- crm_lead_demo.xml
|-- crm_lost_reason_data.xml
|-- crm_stage_data.xml
|-- crm_team_data.xml
|-- crm_team_demo.xml
```

Split them by purpose:

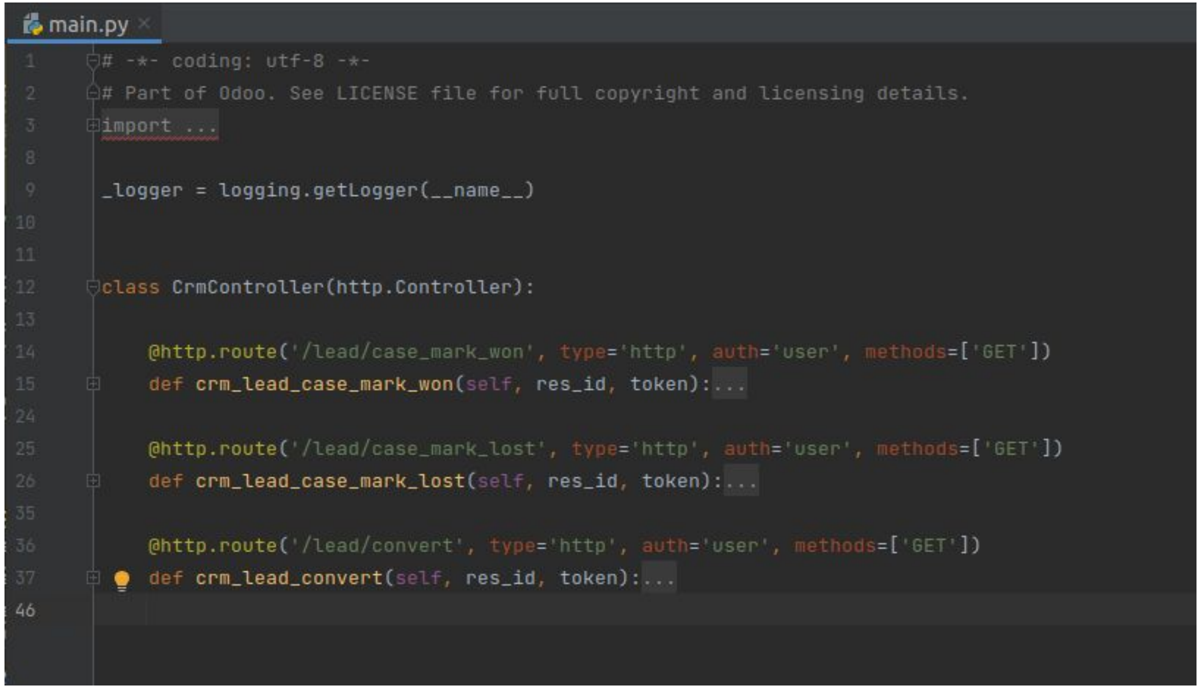
- demo: <model>_demo.xml
- data: <model>_data.xml

01

Module structure

File naming - controller

- outdated: main.py
- now: <module_name>.py
- inherit: <inherited_module_name>.py



main.py

```
# -*- coding: utf-8 -*-
# Part of Odoo. See LICENSE file for full copyright and licensing details.
import ...

_logger = logging.getLogger(__name__)


class CrmController(http.Controller):

    @http.route('/lead/case_mark_won', type='http', auth='user', methods=['GET'])
    def crm_lead_case_mark_won(self, res_id, token):...

    @http.route('/lead/case_mark_lost', type='http', auth='user', methods=['GET'])
    def crm_lead_case_mark_lost(self, res_id, token):...

    @http.route('/lead/convert', type='http', auth='user', methods=['GET'])
    def crm_lead_convert(self, res_id, token):...
```

01

Module structure

File naming - static

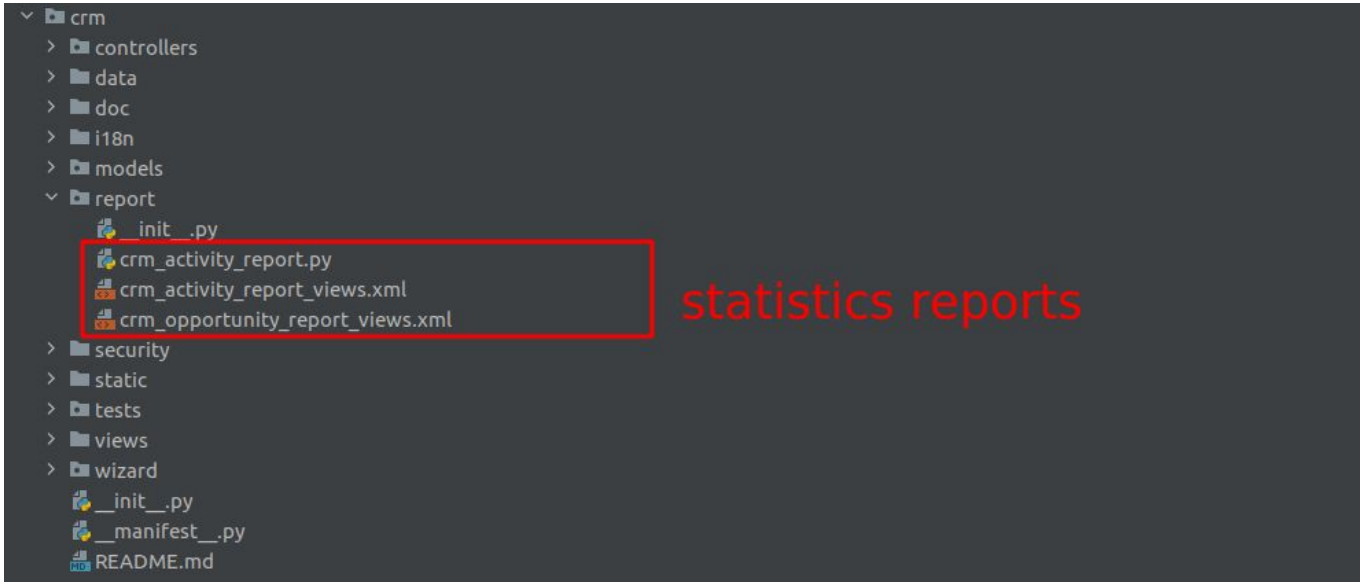
Go to Javascript & CSS section

01

Module structure

File naming - report

Statistics report



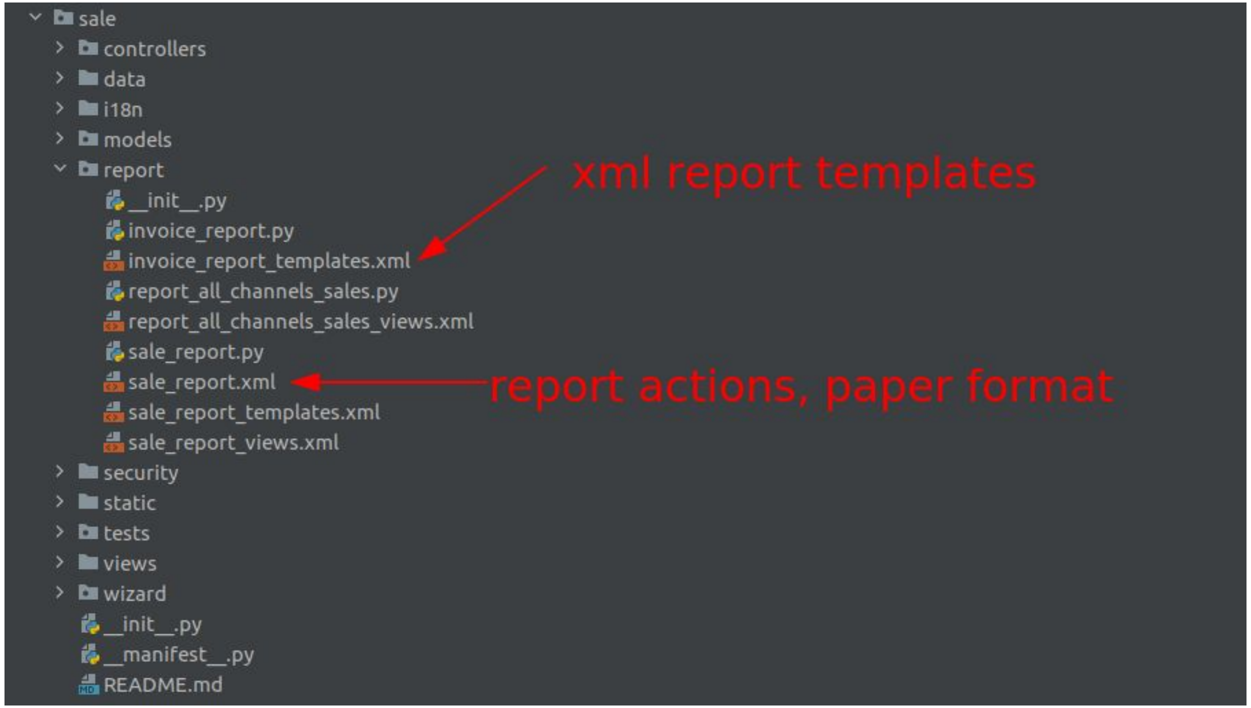
```
crm
  controllers
  data
  doc
  i18n
  models
  report
    __init__.py
    crm_activity_report.py
    crm_activity_report_views.xml
    crm_opportunity_report_views.xml
  security
  static
  tests
  views
  wizard
  __init__.py
  __manifest__.py
  README.md
```

01

Module structure

File naming - report

Printable report



```
sale
  controllers
  data
  i18n
  models
  report
    __init__.py
    invoice_report.py
    invoice_report_templates.xml
    report_all_channels_sales.py
    report_all_channels_sales_views.xml
    sale_report.py
    sale_report.xml
    sale_report_templates.xml
    sale_report_views.xml
  security
  static
  tests
  views
  wizard
  __init__.py
  __manifest__.py
  README.md
```

01

Module structure

File naming

File names should only contain [a-z0-9_] (lowercase alphanumerics and _)

02

Formatting Rules

XML Files - Format

- id before model
- fields: name then eval then others (widgets, options, . . .)
- group records by model except dependencies between action/menu/views
- naming convention (later)

02

Formatting Rules

XML Files - Format

```
<record id="view_id" model="ir.ui.view">
        <field name="name">view.name</field>
        <field name="model">object_name</field>
        <field name="priority" eval="16"/>
        <field name="arch" type="xml">
        <tree>
                <field name="my_field_1"/>
                <field name="my_field_2" string="My Label"
                        widget="statusbar"
                        statusbar_visible="draft,sent,progress,done" />
        </tree>
        </field>
</record>
```

02

Formatting Rules

XML Files - Format

- Syntactic Sugar
    - <menuitem>: ir.ui.menu
    - <template>: arch section of qweb view
    - <report>: report action (old)
    - <act_window>: action window (old)

02

Formatting Rules

XML IDs and Naming - Security, View and Action

- menu: <model_name>_menu
- submenu: <model_name>_menu_do_stuff

```
<!-- menus and sub-menus -->
<menuitem
   id="model_name_menu_root"
   name="Main Menu"
   sequence="5"
/>
<menuitem
   id="model_name_menu_action"
   name="Sub Menu 1"
   parent="module_name.module_name_menu_root"
   action="model_name_action"
   sequence="10"
/>
```

02

Formatting Rules

XML IDs and Naming - Security, View and Action

- menu: <model_name>_menu
- submenu: <model_name>_menu_do_stuff
- view: <model_name>_view_<view_type>

```
<!-- views  -->
<record id="model_name_view_form" model="ir.ui.view">
  <field name="name">model.name.view.form</field>
  ...
</record>

<record id="model_name_view_kanban" model="ir.ui.view">
  <field name="name">model.name.view.kanban</field>
  ...
</record>
```

onnet
BY AHT TECH JSC

02

Formatting Rules

XML IDs and Naming - Security, View and Action

- menu: <model_name>_menu
- submenu: <model_name>_menu_do_stuff
- view: <model_name>_view_<view_type>
- action: <model_name>_action

```
<!-- actions -->
<record id="model_name_action" model="ir.act.window">
  <field name="name">Model Main Action</field>
    ...
</record>

<record id="model_name_action_child_list"
        model="ir.actions.act_window">
  <field name="name">Model Access Children</field>
</record>
```

02

Formatting Rules

XML IDs and Naming - Security, View and Action

- menu: <model_name>_menu
- submenu: <model_name>_menu_do_stuff
- view: <model_name>_view_<view_type>
- action: <model_name>_action
- group: <module_name>_group_<group_name>
- rule: <model_name>_rule_<concerned_group>

```
<!-- actions -->
<record id="model_name_action" model="ir.act.window">
  <field name="name">Model Main Action</field>
    ...
</record>

<record id="model_name_action_child_list"
        model="ir.actions.act_window">
  <field name="name">Model Access Children</field>
</record>
```

02

Formatting Rules

Inheriting XML

name: suffix .inhert.{detail}

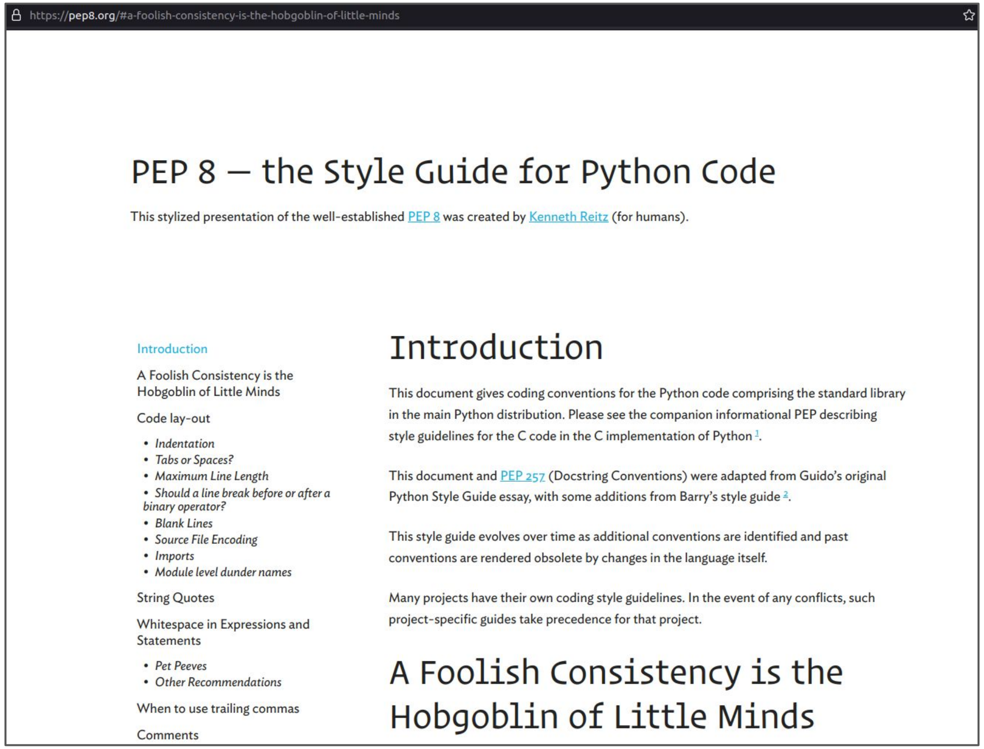
```
<record id="model_view_form" model="ir.ui.view">
  <field name="name">model.view.form.inherit.module2</field>
  <field name="inherit_id" ref="module1.model_view_form"/>
  ...
</record>
<record id="module2.model_view_form" model="ir.ui.view">
  <field name="name">model.view.form.module2</field>
  <field name="inherit_id" ref="module1.model_view_form"/>
  <field name="mode">primary</field>
  ...
</record>
```

03

Python

PEP8 Options

Odoo source code tries to respect Python standard, but some of them can be ignored.



https://pep8.org/#a-foolish-consistency-is-the-hobgoblin-of-little-minds

PEP 8 — the Style Guide for Python Code

This stylized presentation of the well-established PEP 8 was created by Kenneth Reitz (for humans).

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python [1].

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

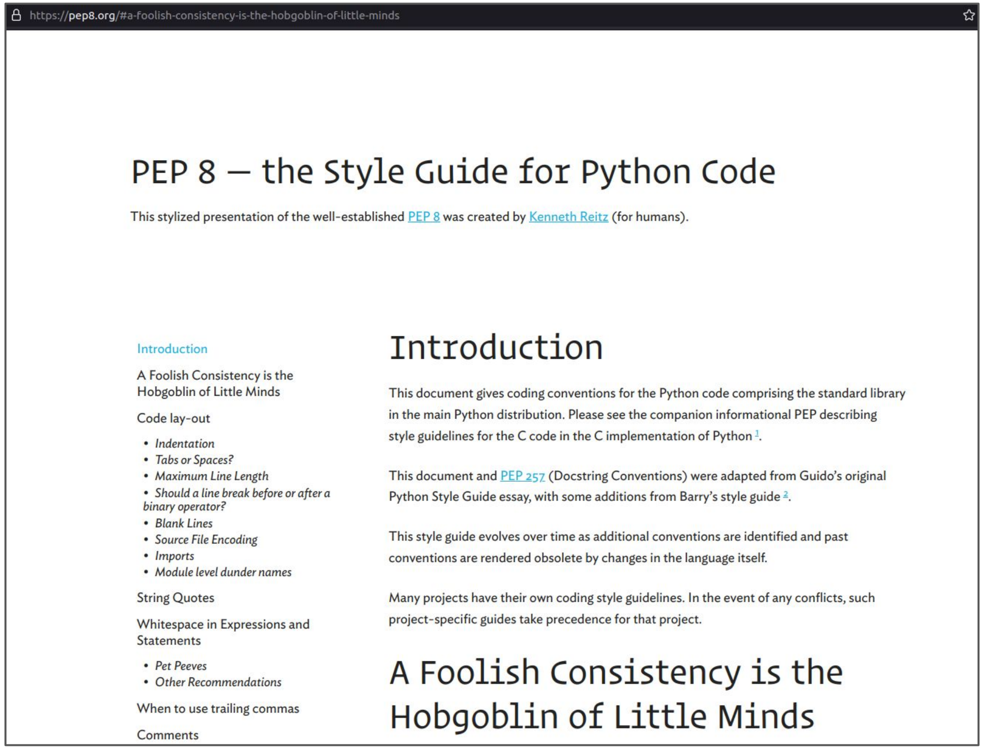
A Foolish Consistency is the Hobgoblin of Little Minds

03

Python

PEP8 Options

Odoo source code tries to respect Python standard, but some of them can be ignored.



https://pep8.org/#a-foolish-consistency-is-the-hobgoblin-of-little-minds

PEP 8 — the Style Guide for Python Code

This stylized presentation of the well-established PEP 8 was created by Kenneth Reitz (for humans).

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python [1].

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

A Foolish Consistency is the Hobgoblin of Little Minds

03

Python

Imports

```
# 1 : imports of python lib
import base64
import re
import time
from datetime import datetime
# 2 : imports of odoo
import odoo
from odoo import api, fields, models, _
from odoo.tools.safe_eval import safe_eval as eval
# 3 : imports from odoo addons
from odoo.addons.website.models.website import slug
```

03

Python

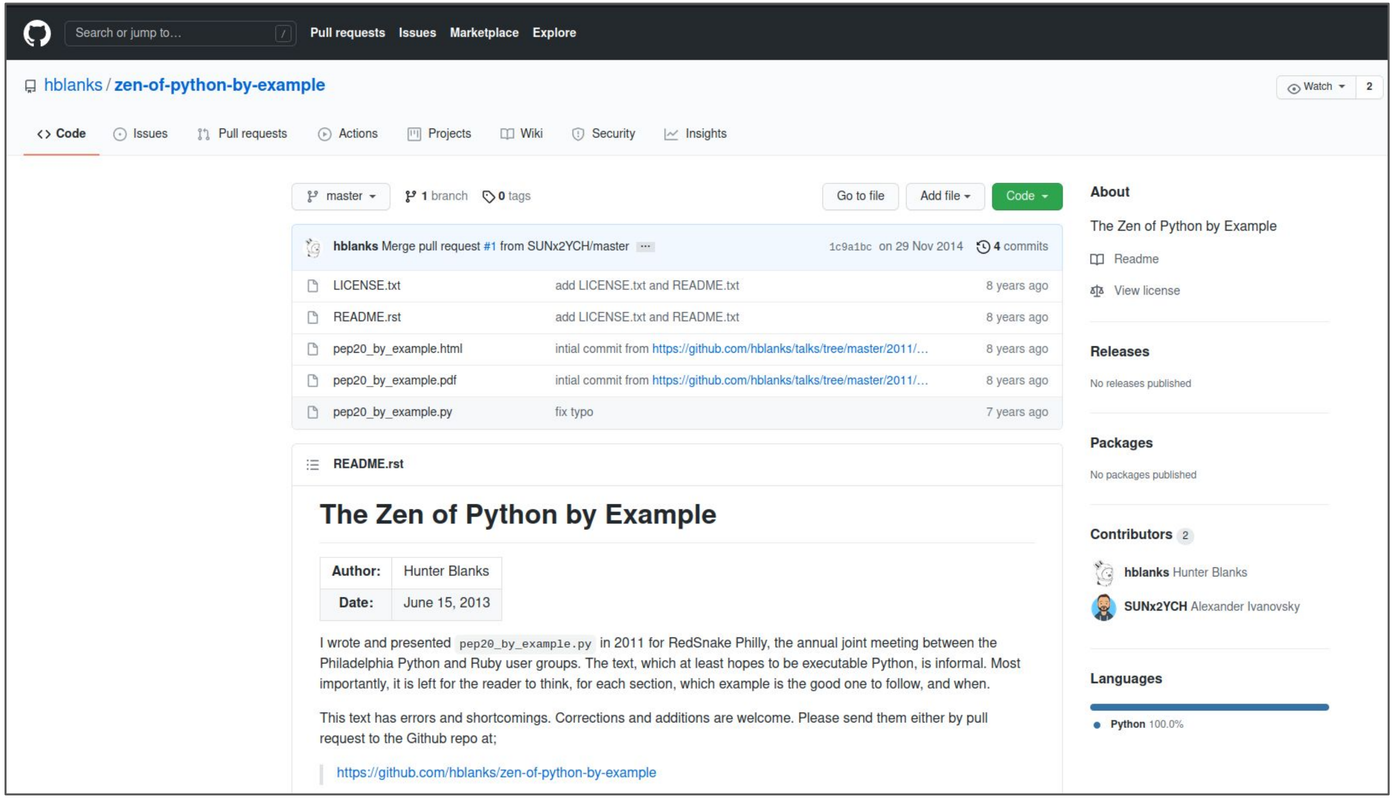
Idiomatics of Programming (Python)

```
>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
...
```

03

Python

Idiomatics of Programming (Python)



hblanks Merge pull request #1 from SUNx2YCH/master ... 1c9a1bc on 29 Nov 2014 4 commits

LICENSE.txt	add LICENSE.txt and README.txt	8 years ago
README.rst	add LICENSE.txt and README.txt	8 years ago
pep20_by_example.html	intial commit from https://github.com/hblanks/talks/tree/master/2011/...	8 years ago
pep20_by_example.pdf	intial commit from https://github.com/hblanks/talks/tree/master/2011/...	8 years ago
pep20_by_example.py	fix typo	7 years ago

README.rst

The Zen of Python by Example

Author:	Hunter Blanks
Date:	June 15, 2013

I wrote and presented `pep20_by_example.py` in 2011 for RedSnake Philly, the annual joint meeting between the Philadelphia Python and Ruby user groups. The text, which at least hopes to be executable Python, is informal. Most importantly, it is left for the reader to think, for each section, which example is the good one to follow, and when.

This text has errors and shortcomings. Corrections and additions are welcome. Please send them either by pull request to the Github repo at;

> https://github.com/hblanks/zen-of-python-by-example

About

The Zen of Python by Example

Readme

View license

Releases

No releases published

Packages

No packages published

Contributors 2

hblanks Hunter Blanks

SUNx2YCH Alexander Ivanovsky

Languages

● Python 100.0%

03

Python

Idiomatics of Programming (Python)

- Use meaningful variable/class/method names
- Useless variable
- Know your builtins
- Use list comprehension, dict comprehension, and basic manipulation using map, filter, sum, . . . They make the code easier to read.
- Collections are booleans too

03

Python

Idiomatics of Programming (Python)

"You can’t learn to write good code only by following the rules.
To learn to write good code you have to write a
shit-metric-ton of bad code."

— Going beyond the idiomatic Python —

04

Programming in Odoo

- Avoid to create generators and decorators
- Use filtered, mapped, sorted, . . . methods to ease code reading and performance.

04

Programming in Odoo

Make your method work in batch

```
@api.depends('user_id')
def _compute_date_open(self):
  for lead in self:
    lead.date_open = fields.Datetime.now() if lead.user_id else False
```

Propagate the context

- Passing parameter in context can have dangerous side-effects.
- If you need to create a key context influencing the behavior of some object, choose a good name, and eventually prefix it by the name of the module to isolate its impact.

04

Programming in Odoo

- Keep it **Simple** and **Stupid**
    - Split the method as soon as it has more than one responsibility
- Never commit the transaction
    - You should NEVER call cr.commit() yourself,
    - ‘UNLESS. . .
- Use translation method correctly

04

Programming in Odoo

Symbols and Conventions - Variables

- model name: singular form
- suffix your variable name with _id or _ids if it contains a record id or list of id

```
Partner = self.env['res.partner']
partners = Partner.browse(ids)
partner_id = partners[0].id
```

- One2Many and Many2Many fields should always have _ids as suffix
- Many2One fields should have _id as suffix

04

Programming in Odoo

Symbols and Conventions - Method Conventions

- compute field: _compute_<field_name>
- onchange method: _onchange_<field_name>
- constraint method: _check_<constraint_name>

Symbols and Conventions - Method Conventions

- compute field: _compute_<field_name>
- onchange method: _onchange_<field_name>
- constrains method: _check_<constrains_name>
- default method: _default_<field_name>
- selection method: _selection_<field_name>
- search method: _search_<field_name>
- action method: prefix action_ and self.ensure_one() at the beginning of the method

Symbols and Conventions - Model attribute order

1. Private attributes (_name, _description, _inherit, …)
2. Default method and _default_get
3. Field declarations
4. Compute, inverse and search methods, same order as field declaration
5. Selection method (method used to return computed values for selection fields)
6. Constrains method (@api.constrains) and onchange method (@api.onchange)
7. CRUD methods (ORM overrides)
8. Action methods
9. other business method

04

Programming in Odoo

Symbols and Conventions - Model attribute order

```
class Event(models.Model):
    # Private attributes
    _name = 'event.event'
    _description = 'Event'

    # Default methods
    def _default_name(self):
        ...

    # Fields declaration
    name = fields.Char(string='Name', default=_default_name)
    event_type = fields.Selection(string="Type", selection='_selection_type')

    # compute and search fields, in the same order of fields declaration
    @api.depends('seats_max', 'registration_ids.state', 'registration_ids.nb_register')
    def _compute_seats(self):
        ...
   @api.model
    def _selection_type(self):
        return []
```

04

Programming in Odoo

Symbols and Conventions - Model attribute order

```
# Constraints and onchanges
@api.constrains('seats_max', 'seats_available')
def _check_seats_limit(self):
    ...
@api.onchange('date_begin')
def _onchange_date_begin(self):
    ...

# CRUD methods (and name_get, name_search, ...) overrides
def create(self, values):
    ...

# Action methods
def action_validate(self):
    self.ensure_one()

# Business methods
def mail_user_confirm(self):
    ...
```

05

Javascript and CSS

Static files organization

- static: all static files in general
    - static/lib: where js libs should be located, in a sub folder.
    - static/src: the generic static source code folder
        - static/src/css: all css files
        - static/src/js
            - static/src/js/tours: end user tour files (tutorials, not tests)
        - static/src/scss: scss files
        - static/src/xml: all qweb templates that will be rendered in JS
    - static/tests: test related files
        - static/tests/tours: tour test files (not tutorials)
    - static/fonts
    - static/img

05

Javascript and CSS

Javascript coding guideline

- use strict; is recommended for all javascript files
- Use a linter (jshint, …)
- Never add minified Javascript Libraries
- Variables and functions should be camelcased (myVariable) instead of snakecased (my_variable)
- Name all entities exported by a JS module.

```
// Instead of
   return Widget.extend({
       // ...
    });

// you should use
   var MyWidget = Widget.extend({
       // ...
    });
    return MyWidget;
```

05

Javascript and CSS

Javascript coding guideline

- use strict; is recommended for all javascript files
- Use a linter (jshint, …)
- Never add minified Javascript Libraries
- Variables and functions should be camelcased (myVariable) instead of snakecased (my_variable)
- Name all entities exported by a JS module.
- Use strict comparisons (=== instead of ==)
- strings: double quotes for all textual strings (such as "Hello"), and single quotes for all other strings, such as a css selector '.o_form_view'
- Write unit tests
- Always use this._super.apply(this, arguments);
- Document every functions and every files, with the JSDoc style (see http://usejsdoc.org/)

05

Javascript and CSS

Javascript coding guideline

```
/**
 * When a save operation has been confirmed from the model, this method is
 * called.
 *
 * @private
 * @override method from field manager mixin
 * @param {string} id
 */
_confirmSave: function (id) {
```

See more at Odoo wiki

06

Git

Commit message structure

[TAG] module: describe your change in a short sentence (ideally < 50 chars)

Long version of the change description, including the rationale for the change, or a summary of the feature being introduced.

Please spend a lot more time describing WHY the change is being done rather than WHAT is being changed. This is usually easy to grasp by actually reading the diff. WHAT should be explained only if there are technical choices or decision involved. In that case explain WHY this decision was taken.

End the message with references, such as task or bug numbers, PR numbers, and OPW tickets, following the suggested format:
task-123 (related to task)
Fixes #123 (close related issue on Github)
Closes #123 (close related PR on Github)
opw-123 (related to ticket)

06

Git

Tag and module name

- **[FIX]** for bug fixes: mostly used in stable version but also valid if you are fixing a recent bug in development version;
- **[REF]** for refactoring: when a feature is heavily rewritten;
- **[ADD]** for adding new modules;
- **[REM]** for removing resources: removing dead code, removing views, removing modules, …;
- **[REV]** for reverting commits: if a commit causes issues or is not wanted reverting it is done using this tag;
- **[MOV]** for moving files: use git move and do not change content of moved file otherwise Git may loose track and history of the file; also used when moving code from one file to another;
- **[REL]** for release commits: new major or minor stable versions;
- **[IMP]** for improvements: most of the changes done in development version are incremental improvements not related to another tag;
- **[MERGE]** for merge commits: used in forward port of bug fixes but also as main commit for feature involving several separated commits;
- **[CLA]** for signing the Odoo Individual Contributor License;
- **[I18N]** for changes in translation files;

After tag comes the modified module name. Use the technical name as functional name may change with time. If several modules are modified, list them or use various to tell it is cross-modules.

06

Git

Commit message header

A meaningful commit message header

Self explanatory and include the reason behind the change.

Try to limit the header length to about 50 characters for readability.

Commit message header should make a valid sentence once concatenated with if applied, this commit will <header>

06

Git

Commit message full description

Specify the part of the code impacted by your changes (module name, lib, transversal object, …) and a description of the changes.

Explain WHY you are modifying code

Avoid commits which simultaneously impact multiple modules

Don’t hesitate to be a bit verbose.

You spend several hours, days or weeks working on meaningful features. Take some time to calm down and write clear and understandable commit messages.

If you are working on a task that lacks purpose and specifications please consider making them clear before continuing.

Q&A