



ENTERPRISE RESOURCE PLANNING: ODOO

Présenté par : - REMACI ZEYNEB YASMINA
- OUADAH AMINA

FONCTIONS ET MÉTHODES

- L'acronyme ORM signifie "Object-Relational Mapping". C'est une technique de programmation qui permet de faire le lien entre les objets en programmation orientée objet et les données stockées dans une base de données relationnelle.
- L'ORM d'Odoo permet de simplifier la gestion et la manipulation des données dans les applications Odoo en utilisant des objets Python pour représenter les enregistrements dans la base de données.
- Le traitement des données s'effectue à travers des fonctions python.
- La déclaration des fonctions est faite avec le paramètre « **self** » suivie par les paramètres spécifiques.
- « **self** » est une variable spéciale qui fait référence à l'instance de l'objet actuellement en cours de traitement.

API (DÉCORATEUR)

```
from odoo import models, fields, api
```

@api.multi : Décore les méthodes où « self » est un ou plusieurs enregistrements.

@api.one: Décore les méthodes où « self » est un et un seul enregistrement.

@api.onchange('field_name'): Décore les méthodes de changement des champs « onchange ».

@api.depends('field_name'): Utilisé dans les méthodes « **compute** » pour spécifier les dépendances du champ calculé.

@api.constraints('field_name'): Décore les méthodes de vérification des contraintes.

EXEMPLES

```
@api.constrains('date_debut')
def _check_date_debut(self):
    for record in self:
        if record.date_debut < fields.Date.today():
            raise ValidationError("La date de début ne peut pas être définie dans le passé.")
```

EXEMPLES

Gestion de bibliothèque

Auteur livre emprunt emprunteur

emprunt / Nouveau

Sauvegarder Annuler

Date début 11/10/2023

Emprunteur Remaci

Livres

livre

Ajouter une ligne

Odoo Server Error - Erreur de validation

Erreur lors de la validation de la contrainte

La date de début ne peut pas être définie dans le passé.

Ok

CHAMPS CALCULABLE (COMPUTE) VS ONCHANGE

- Dans Odoo, il existe deux manières de calculer automatiquement la valeur d'un champ. Nous pouvons utiliser le décorateur "**onchange**" ou le paramètre de champ "**compute**".
- Utilisez le décorateur "**onchange**" si vous souhaitez que la valeur d'un champ change automatiquement lorsque la valeur d'un autre champ est modifiée par l'utilisateur.
- Le champ que nous avons calculé avec le décorateur "**onchange**" peut toujours être modifié manuellement par l'utilisateur.

EXAMPLES

```
@api.onchange('date_naissance')
def calculer_age(self):
    for rec in self:
        if rec.date_naissance:
            date_n = rec.date_naissance
            d2 = date.today()
            rd = relativedelta(d2, date_n)
            rec.age = rd.years
```

EXEMPLES

Nom

Mohamed

Nationalité

Alg

Prénom

Amine

Sexe

Femme

Date de naissance

12/12/2004

Age

18

CHAMPS CALCULABLE (COMPUTE) VS ONCHANGE

De plus, nous pouvons également utiliser le paramètre de champ "**compute**" pour calculer automatiquement la valeur d'un champ.


Le champ calculable:

1. Il est déterminé en se basant sur des champs existants.
2. Le champ est spécifié avec l'attribut « compute ».
3. La valeur de l'attribut « compute » est le nom de la fonction python dans la quelle le champs est calculer.
4. Le champ que nous avons marqué avec le paramètre de champ "compute" sera automatiquement défini en lecture seule.
5. Si nous voulons que la valeur du champ résultat change immédiatement sans avoir à cliquer sur le bouton "Enregistrer", vous pouvez également utiliser le décorateur « depends ».

EXEMPLES

```
duree=fields.Char(string="Durée", compute='compute_duree')
```

```
@api.depends('date_debut', 'date_fin')
def compute_duree(self):
    if self.date_debut and self.date_fin:
        self.duree = (self.date_fin - self.date_debut).days
```

| | | | |
|------------|---|---|---|
| Date début | <input type="text" value="21/10/2023"/> | Date fin | <input type="text" value="31/10/2023"/> |
| Emprunteur | <input type="text" value="Remaci"/> |  Rendu | <input type="text" value="Non"/> |
| Durée | 10 | | |

QUELQUE FONCTION D'ORM

self.env['object.name']

- Retourne l'objet 'object.name'

self.env['object.name'].create({dict})

- Crée un enregistrement dans l'objet 'object.name' avec les valeurs du dictionnaire.

self.env['object.name'].write(id,{dict})

- Modifier l'enregistrement avec l'identificateur 'id' de l'objet 'object.name' .

self.env['object.name'].unlink(id)

- Supprimer l'enregistrement avec l'identificateur 'id' de l'objet 'object.name' .

self.env.ref('Module.External_id')

- Retourne l'enregistrement qui a comme id externe 'External_id' . (celui du XML)

self.env['object.name'].search([Domaine])

- Retourne l'enregistrement (1 ou n) qui satisfait la condition du domaine.

self.env['object.name'].browse(id)

- Retourne l'enregistrement de l'objet 'object.name' avec l'identificateur 'id'

QUELQUE FONCTION D'ORM

```
@api.depends('livre_ids')
def compute_livres(self):
    livre_ids = self.env['livre'].search([('nbr_pages', '>', 500)])
    self.nbr_livre = len(livre_ids)
```