

Entretien Odoo

1. Problèmes de codage rencontrés (avec solutions)

Count IP Addresses (Easy)

Compter le nombre d'adresses IP entre deux adresses données.

```
def ips_between(start, end):
    def ip_to_int(ip):
        parts = list(map(int, ip.split('.')))
        return (parts[0] << 24) + (parts[1] << 16) + (parts[2] << 8) + parts[3]
    return ip_to_int(end) - ip_to_int(start)

# Exemple
print(ips_between("10.0.0.0", "10.0.0.50")) # 50
print(ips_between("20.0.0.10", "20.0.1.0")) # 246
```

Your Order Please (Easy)

Trier les mots d'une phrase selon le chiffre qu'ils contiennent.

```
def order(sentence):
    if not sentence:
        return ""
    words = sentence.split()
    return ' '.join(sorted(words, key=lambda w: next(c for c in w if c.isdigit()))))

# Exemple
print(order("is2 Thi1s T4est 3a")) # "Thi1s is2 3a T4est"
```

Guess Number Higher or Lower (Easy) — LeetCode 374

Recherche binaire simple.

```
def guessNumber(n):
    lo, hi = 1, n
    while lo <= hi:
        mid = (lo + hi) // 2
        result = guess(mid) # -1 = trop haut, 1 = trop bas, 0 = trouvé
        if result == 0:
            return mid
        elif result == -1:
            hi = mid - 1
        else:
            lo = mid + 1
```

MinStack (Medium) — LeetCode 155

Stack qui supporte `push`, `pop`, `top` et `getMin` en $O(1)$.

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, val):
        self.stack.append(val)
        # on empile le min courant
        self.min_stack.append(min(val, self.min_stack[-1] if self.min_stack else val))

    def pop(self):
        self.stack.pop()
        self.min_stack.pop()

    def top(self):
        return self.stack[-1]
```

```
def getMin(self):  
    return self.min_stack[-1]
```

Evaluate Reverse Polish Notation (Medium) — LeetCode 150

Évaluer une expression en notation polonaise inversée.

```
def evalRPN(tokens):  
    stack = []  
    ops = {  
        '+': lambda a, b: a + b,  
        '-': lambda a, b: a - b,  
        '*': lambda a, b: a * b,  
        '/': lambda a, b: int(a / b), # troncature vers zéro  
    }  
    for t in tokens:  
        if t in ops:  
            b, a = stack.pop(), stack.pop()  
            stack.append(ops[t](a, b))  
        else:  
            stack.append(int(t))  
    return stack[0]  
  
# Exemple  
print(evalRPN(["2", "1", "+", "3", "*"])) # 9 → ((2+1)*3)
```

Syntax Scoring — Advent of Code 2021 Jour 10 (Medium)

Trouver les caractères illégaux dans des lignes de parenthèses/brackets.

```
def syntax_score(lines):  
    pairs = {'(': ')', '[': ']', '{': '}', '<': '>'}  
    points = {')': 3, ']': 57, '}': 1197, '>': 25137}  
    score = 0
```

```

for line in lines:
    stack = []
    for ch in line:
        if ch in pairs:
            stack.append(pairs[ch])
        elif not stack or stack.pop() != ch:
            score += points[ch]
            break
    return score

```

Partie 2 : compléter les lignes incomplètes

```

def completion_score(lines):
    pairs = {'(': ')', '[': ']', '{': '}', '<': '>'}
    comp_points = {')': 1, ']': 2, '}': 3, '>': 4}
    scores = []

    for line in lines:
        stack = []
        corrupt = False
        for ch in line:
            if ch in pairs:
                stack.append(pairs[ch])
            elif not stack or stack.pop() != ch:
                corrupt = True
                break
        if not corrupt and stack:
            s = 0
            for ch in reversed(stack):
                s = s * 5 + comp_points[ch]
            scores.append(s)

    scores.sort()
    return scores[len(scores) // 2] # score médian

```

Dumbo Octopus — Advent of Code 2021 Jour 11 (Medium)

Simulation de grille avec réaction en chaîne (flash quand énergie > 9).

```

def simulate_octopus(grid, steps=100):
    rows, cols = len(grid), len(grid[0])
    total_flashes = 0

    def neighbors(r, c):
        for dr in [-1, 0, 1]:
            for dc in [-1, 0, 1]:
                if dr == 0 and dc == 0: continue
                nr, nc = r + dr, c + dc
                if 0 <= nr < rows and 0 <= nc < cols:
                    yield nr, nc

    for step in range(steps):
        # 1. incrémenter tout
        for r in range(rows):
            for c in range(cols):
                grid[r][c] += 1

        # 2. flash en chaîne
        flashed = set()
        changed = True
        while changed:
            changed = False
            for r in range(rows):
                for c in range(cols):
                    if grid[r][c] > 9 and (r, c) not in flashed:
                        flashed.add((r, c))
                        changed = True
                        for nr, nc in neighbors(r, c):
                            grid[nr][nc] += 1

        # 3. reset les flashés à 0
        total_flashes += len(flashed)
        for r, c in flashed:
            grid[r][c] = 0

    return total_flashes

```

Rectangle Partition (Hard — CodinGame)

Compter les rectangles formés par des lignes horizontales et verticales dans un grand rectangle.

```
def rectangle_partition(w, h, x_lines, y_lines):  
    # Ajouter les bords  
    xs = [0] + sorted(x_lines) + [w]  
    ys = [0] + sorted(y_lines) + [h]  
  
    # Calculer tous les segments possibles sur chaque axe  
    from collections import Counter  
    x_gaps = Counter()  
    y_gaps = Counter()  
  
    for i in range(len(xs)):  
        for j in range(i + 1, len(xs)):  
            x_gaps[xs[j] - xs[i]] += 1  
  
    for i in range(len(ys)):  
        for j in range(i + 1, len(ys)):  
            y_gaps[ys[j] - ys[i]] += 1  
  
    # Un carré existe quand un segment X == un segment Y  
    # Pour rectangles : compter toutes les combinaisons  
    count = 0  
    for size, xc in x_gaps.items():  
        if size in y_gaps:  
            count += xc * y_gaps[size]  
    return count # pour les carrés ; adapter pour tous les rectangles
```

2. SQL & Bases de données

SQL vs NoSQL

	SQL	NoSQL
Structure	Tables, schéma fixe	Documents, clé-valeur, graphe...

	SQL	NoSQL
Schéma	Rigide, défini à l'avance	Flexible, schemaless
Relations	Jointures natives	Dénormalisation / embedding
Scaling	Vertical (scale up)	Horizontal (scale out)
ACID	Oui	Souvent eventual consistency
Exemples	PostgreSQL, MySQL	MongoDB, Redis, Cassandra
Quand ?	Données structurées, relations complexes	Gros volumes, schéma variable

Jointures SQL

```
-- INNER JOIN : seulement les correspondances
SELECT e.name, d.name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.id;

-- LEFT JOIN : tous les employés, même sans département
SELECT e.name, d.name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.id;

-- RIGHT JOIN : tous les départements, même sans employés
SELECT e.name, d.name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.id;

-- FULL OUTER JOIN : tout de chaque côté
SELECT e.name, d.name
FROM employees e
FULL OUTER JOIN departments d ON e.dept_id = d.id;
```

Index

C'est quoi ? Structure de données (souvent B-tree) qui accélère les recherches.

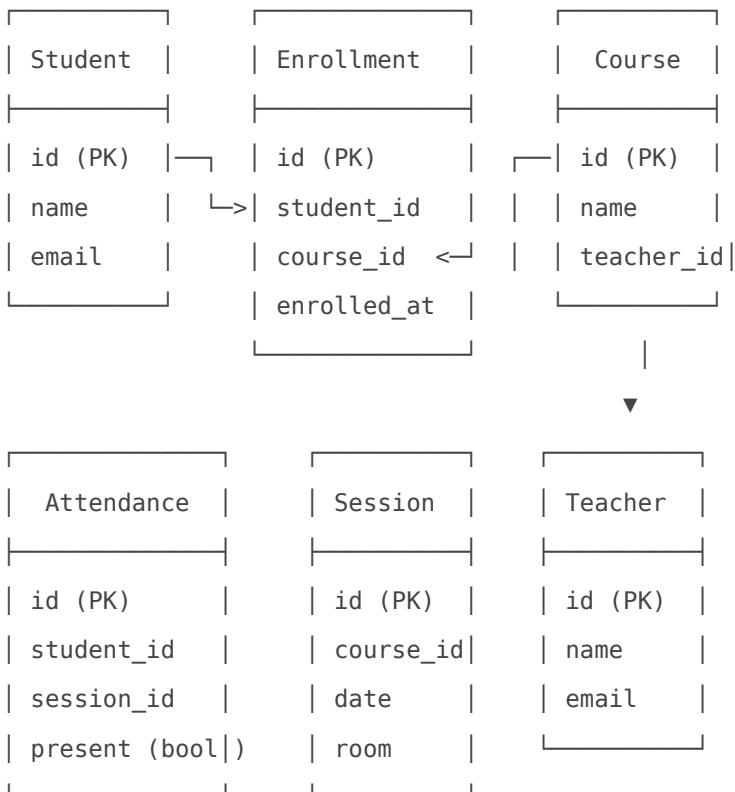
```
CREATE INDEX idx_email ON users(email);
```

- **Avantage** : SELECT/WHERE beaucoup plus rapides
- **Inconvénient** : ralentit INSERT/UPDATE, consomme de l'espace
- **Quand ?** : colonnes fréquemment filtrées, clés étrangères, colonnes dans ORDER BY
- **Éviter sur** : petites tables, colonnes rarement utilisées dans WHERE

Salaire moyen maximum (question classique)

```
-- Département avec le salaire moyen le plus élevé
SELECT d.name, AVG(e.salary) AS avg_salary
FROM employees e
JOIN departments d ON e.dept_id = d.id
GROUP BY d.name
ORDER BY avg_salary DESC
LIMIT 1;
```

Modélisation — Exemple : plateforme de cours



Relations :

- Student ↔ Course : Many-to-Many (via Enrollment)

- Course → Teacher : Many-to-One
- Session → Course : Many-to-One
- Attendance → Student + Session : Many-to-One chacune

Normalisation express

Forme	Règle
1NF	Pas de valeurs multiples dans une cellule
2NF	1NF + chaque colonne non-clé dépend de toute la clé primaire
3NF	2NF + pas de dépendances transitives ($A \rightarrow B \rightarrow C$ interdit)

3. Questions techniques — Réponses

`===` vs `==` en JavaScript

```
// == : comparaison avec coercion de type
0 == ""          // true  (les deux deviennent 0)
null == undefined // true
"5" == 5         // true

// === : comparaison stricte (type + valeur)
0 === ""         // false
null === undefined // false
"5" === 5        // false
```

Règle : toujours utiliser `===` sauf cas très spécifique.

`static`

```
class MathUtils:
  counter = 0 # variable de classe (partagée)

  @staticmethod
```

```
def add(a, b): # pas besoin d'instance
    return a + b

@classmethod
def increment(cls):
    cls.counter += 1

# Appel sans instancier
MathUtils.add(2, 3) # 5
```

En résumé : `static` = appartient à la classe, pas à l'instance. Partagé entre tous les objets.

Polymorphisme

```
class Animal:
    def speak(self):
        raise NotImplementedError

class Dog(Animal):
    def speak(self):
        return "Woof"

class Cat(Animal):
    def speak(self):
        return "Meow"

# Même interface, comportement différent
for animal in [Dog(), Cat()]:
    print(animal.speak()) # Woof, Meow
```

Deux types :

- **Compile-time** (surcharge / overloading) : même nom, signatures différentes
- **Runtime** (héritage / overriding) : sous-classe redéfinit la méthode parente

Classe abstraite vs Interface

	Classe abstraite	Interface
--	------------------	-----------

Instanciation	Non	Non
Méthodes concrètes	Oui (mix abstrait + concret)	Non (que des signatures)*
Héritage	Simple (1 seule parent)	Multiple
Attributs	Oui	Non (sauf constantes)
Usage	Partager du code commun	Définir un contrat

*En Python, pas de distinction formelle : on utilise `ABC` pour les deux.

```
from abc import ABC, abstractmethod

class Shape(ABC):          # classe abstraite
    @abstractmethod
    def area(self):
        pass

    def describe(self):     # méthode concrète partagée
        return f"Area = {self.area()}"

class Circle(Shape):
    def __init__(self, r):
        self.r = r
    def area(self):
        return 3.14 * self.r ** 2
```

Git rebase vs merge

```
# merge : crée un commit de fusion, préserve l'historique
git checkout main
git merge feature          # historique non-linéaire, commit de merge

# rebase : réécrit l'historique, résultat linéaire
git checkout feature
git rebase main            # rejoue les commits de feature sur main
git checkout main
git merge feature          # fast-forward, historique propre
```

	merge	rebase
Historique	Non-linéaire, avec commit de merge	Linéaire, propre

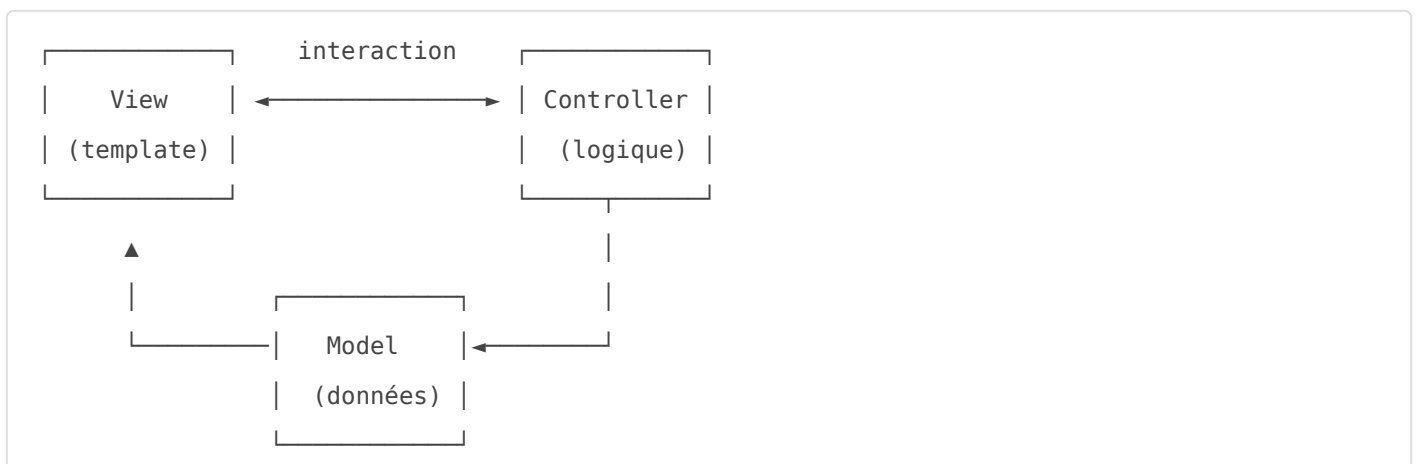
	merge	rebase
Sûr ?	Oui (jamais destructif)	Danger si branche partagée
Quand ?	Branches partagées, PR	Nettoyage local avant merge

Règle d'or : ne jamais rebase une branche publique.

Que se passe-t-il quand on tape une URL ?

1. **DNS** : le navigateur résout le domaine → adresse IP
2. **TCP** : connexion en 3 étapes (SYN, SYN-ACK, ACK)
3. **TLS** : si HTTPS, handshake pour chiffrer
4. **HTTP** : envoi de la requête GET
5. **Serveur** : traite la requête, renvoie HTML
6. **Rendu** : parsing HTML → DOM, CSS → CSSOM, JS exécuté, layout, paint

Architecture MVC



- **Model** : données + logique métier (ex : `sale.order` dans Odoo)
- **View** : affichage (templates QWeb dans Odoo)
- **Controller** : reçoit les requêtes, orchestre Model ↔ View

Odoo utilise une variante MVC : les modèles ORM sont le Model, les vues XML le View, et les contrôleurs HTTP le Controller.

Connexion Peer-to-Peer

- Pas de serveur central : chaque nœud est client ET serveur
- **NAT traversal** : techniques comme STUN/TURN pour traverser les pare-feux

- **Exemples** : BitTorrent, WebRTC (appels vidéo)
 - **Avantage** : pas de point unique de défaillance, décentralisé
 - **Inconvénient** : découverte de pairs complexe, sécurité plus difficile
-

4. Checklist dernière minute

- ☐ Résoudre 2-3 problèmes easy + 2 medium sur LeetCode
 - ☐ Écrire 5 requêtes SQL de tête (JOIN, GROUP BY, HAVING, sous-requête)
 - ☐ Dessiner un schéma de données en 10 min sur papier
 - ☐ Réviser OOP : héritage, polymorphisme, encapsulation, abstraction
 - ☐ Préparer le pitch "Parlez-moi de vous" (2 min max)
 - ☐ Vérifier qu'on sait expliquer chaque projet de son CV
 - ☐ Laptop chargé + stylo + papier si entretien en personne
-

Revision #2

Created 22 April 2026 22:45:17 by Hugo

Updated 22 April 2026 22:55:07 by Hugo