Exercice 1 (6 points)

Correction

| Q | N | Thème | Réponse |
| --- | --- | --- | --- |
| 1 | 1 | Arbre binaire | Oui on peut l’identifier, il s’agit d’un sorbier. |
| 2 | 1 | Arbre binaire | Non on ne peut pas l’identifier, aucun végétal connu n’a ces caractéristiques. |
| 3 | 1 | Programmation Orientée Objet | 1 sorbier = Feuille\_resultat(["Sorbier"]) 2 robinier\_noyer = Feuille\_resultat(["Robinier", "Noyer"]) 3 feuille\_vide1 = Feuille\_resultat([]) 4 feuille\_vide2 = Feuille\_resultat([]) 5 n\_bord = Noeud("Bord denté ?", sorbier, robinier\_noyer) 6 n\_alt = Noeud("Alternées ?", n\_bord, feuille\_vide1) 7 arbre\_2 = Noeud("Simples ?", feuille\_vide2, n\_alt) |
| 4 | 1 | Programmation Orientée Objet | Pour la classe Noeud :  1 def est\_resultat(self): 2 return False |
| 5 | 1 | Programmation Orientée Objet | Pour la classe Feuille\_resultat :  1 def est\_resultat(self): 2 return True |
| 6 | 1 | Programmation Orientée Objet | Pour la classe Feuille\_resultat :  1 def nb\_vegetaux(self): 2 return len(self.vegetaux) |
| 7 | 2 | Algorithmes sur les arbres binaires | Pour la classe Feuille\_resultat :  1 def nb\_vegetaux(self): 2 return self.sioui.nb\_vegetaux() + self.sinon.nb\_vegetaux() |
| 8 | 1 | Programmation Orientée Objet | Pour la classe Feuille\_resultat :  1 def liste\_questions(self): 2 return [] |
| 9 | 2 | Algorithmes sur les arbres binaires | Pour la classe Feuille\_resultat :  1 def liste\_questions (self): 2 return [self.question] + self.sioui.liste\_questions() + self.sinon.liste\_questions () |
| 10 | 2 | Algorithmes sur les arbres binaires | 1 def est\_bien\_renseigne (dico\_vegetal, arbre) : 2 liste\_q = arbre.liste\_questions() 3 for q in liste\_q : 4 if not q in dico\_vegetal : 5 return False 6 return True |
| 11 | 3 | Algorithmes sur les arbres binaires | 1 def identifier\_vegetaux(arbre, dico\_vegetal): 2 if arbre.est\_resultat(): 3 return arbre.vegetaux 4 if dico\_vegetal[arbre.question]: 5 return identifier\_vegetaux(arbre.sioui, dico\_vegetal) 6 else: 7 return identifier\_vegetaux(arbre.sinon, dico\_vegetal) |

Exercice 2 (6 points)

Correction

| Q | N | Thème | Réponse |
| --- | --- | --- | --- |
| 1 | 1 | Programmation Python, POO | 1 def passer\_transit(self): 2 self.etat = 'transit' |
| 2 | 2 | Programmation Python | 1 def ajouter\_colis(liste, colis): 2 if colis.poids <= 25: 3 liste.append(colis) 4 else: 5 print('Dépassement du poids maximal autorisé') |
| 3 | 1 | Parcours séquentiel d'un tableau | 1 def nb\_colis(liste): 2 return len(liste) |
| 4 | 2 | Parcours séquentiel d'un tableau | 1 def poids\_total(liste): 2 total = 0  3 for c in liste : 4 total = total + c.poids  5 return total |
| 5 | 2 | Parcours séquentiel d'un tableau, POO | 1 def liste\_colis\_etat(liste, statut): 2 resultat = [] 3 for c in liste: 4 if c.etat == statut: 5 resultat.append(c) 6 return resultat |
| 6 | 1 | Tris par sélection | Le tri par sélection de coût quadratique. |
| 7 | 1 | Tris | Le tri par insertion (coût quadratique) ou le tri fusion (coût *n log n*), ou n’importe quel autre tri. |
| 8 | 3 | Algorithme glouton, récursivité | 1 def chargement\_glouton(liste, rang, capacite): 2 if rang == len(liste): 3 return []  4 elif liste[rang].poids <= capacite:  5 return [liste[rang]] + chargement\_glouton(liste, rang+1, capacite - liste[rang].poids)  6 else: 7 return chargement\_glouton(liste, rang + 1, capacite) |
| 9 | 2 | recursivité | La récursivité utilise une pile d’appels qui est un espace mémoire particulièrement limité, cela génère rapidement des débordements de capacité. |
| 10 | 3 | Algorithme glouton | 1 def chargement\_glouton2(liste, capacite):  2 #initialisation des variables  3 poids\_total = 0   4 colis\_a\_charger = []   5 # ajout si possible du prochain colis  6 for colis in liste\_trie:  7 if poids\_total + colis.poids <= capacite :   8 colis\_a\_charger.append(colis)   9 poids\_total = poids\_total + colis.poids  10 return colis\_a\_charger |

Exercice 3 (8 points)

Correction

| Q | N | Thème | Réponse |
| --- | --- | --- | --- |
| 1 | 1 | bases\_de\_donnees | L’attribut annee devrait être un entier. |
| 2 | 2 | bases\_de\_donnees | Pour être cohérent, on pourrait contraindre ses valeurs de 2007 à 2025 qui correspondrait aux dates de naissances d’enfants (). |
| 3 | 1 | bases\_de\_donnees | L’attribut parent en tant que clef étrangère suit une contrainte de référence. |
| 4 | 1 | bases\_de\_donnees | L’attribut tel permet d’identifier de façon unique un parent et peut donc être une clef primaire. (Un nom n’est pas forcément unique car il y a des homonymes, et le code postal n’est pas unique à une personne non plus). |
| 5 | 2 | bases\_de\_donnees | Cette requête lève une erreur car les enfants Maya (17) et Rachelle (23) ont pour parent le parent 33600781122. |
| 6 | 1 | bases\_de\_donnees | INSERT INTO parent VALUES ('Bauges', 33619782812, 73340); UPDATE enfant  SET num\_parent = 33619782812  WHERE num\_parent = 33600782812; DELETE FROM parent WHERE tel = 33600782812; |
| 7 | 1 | bases\_de\_donnees | Le résultat est Nakamura, Hawa, Kian, Adrien |
| 8 | 1 | bases\_de\_donnees | SELECT prenom FROM enfant WHERE num\_parent = 33619861122 ORDER BY prenom; |
| 9 | 2 | bases\_de\_donnees | SELECT id, prenom FROM enfant JOIN parent ON num\_parent = tel WHERE codep = 38520; |
| 10 | 1 | graphe | La mésentente entre deux enfants est une relation symétrique, ainsi si l’enfant A est en mésentente avec l’enfant B il en sera de même de l’enfant B avec l’enfant A. Un graphe non-orienté suffit donc. |
| 11 | 1 | graphe | Figure 4. Graphe g2 |
| 12 | 1 | langages\_programmation | À l’aide d’une boucle ou bien 1 def degre(g, s): 2 return len(g[s]) |
| 13 | 3 | algorithmique | 4 for i in range(1, n):  5 sommet\_courant = sommets[i]  6 j = i-1  7 while j >= 0 and degre(g, sommets[j]) < degre(g, sommet\_courant):   8 sommets[j+1] = sommets[j]   9 j = j - 1 10 sommets[j+1] = sommet\_courant |
| 14 | 2 | algorithmique | Il s’agit d’un tri par insertion de coût d’exécution temporel quadratique. |
| 15 | 2 | graphe | Figure 5. Graphe g1 coloré |
| 16 | 2 | algorithmique | 1 def colorer\_graphe(g, dc): 2 # Pré-condition : les clés de dc sont les sommets de g, et les valeurs de dc sont à -1 3 for s in dc: 4 couleur = plus\_petite\_couleur\_hors\_voisins(g, dc, s)  5 dc[s] = couleur |
| 17 | 3 | algorithmique | 1 def welsh\_powell(g): 2 # initialisation à -1 pour tous les sommets dans le dictionnaire dc 3 dc = {s: -1 for s in g}  4 # coloration en suivant l'approche de Welsh-Powell 5 for s in sommets\_tries(g):  6 couleur = plus\_petite\_couleur\_hors\_voisins(g, dc, s)  7 dc[s] = couleur  8 return dc |