# Exercice 1 (6 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | cours | Identification unique |
| 2 | 2 | cours | On n’aurait pas pu stocker le fait que deux équipes se soient rencontrées deux fois avec le même score final. |
| 3 | 1 | requête d'interrogation sur une table | Henri ; Laure ; Brigitte ; Laure |
| 4 | 1 | requête d'interrogation sur une table | SELECT DISTINCT prenom FROM joueur WHERE ann\_naiss < 1985 |
| 5 | 1 | écriture d'une requête d'interrogation sur une table | SELECT nom, ann\_naiss, num\_port FROM joueur  WHERE commune = 'Bois-Plage'; |
| 6 | 2 | écriture d'une requête d'interrogation sur deux tables | SELECT joueur.nom, joueur.prenom FROM joueur  JOIN equipe ON equipe.j\_1 = joueur.id\_joueur  WHERE equipe.nom = 'Les Kangourous'; |
| 7 | 1 | écriture d'une requête de mise à jour | UPDATE equipe SET points = 5  WHERE nom = 'Volley Warriors'; |
| 8 | 1 | écriture d'une requête de mise à jour | DELETE FROM joueur WHERE id\_joueur = 35; |
| 9 | 1 | explication d'une requête | SELECT id\_match FROM match  WHERE eq\_1 = 12 OR eq\_2 = 12; |
| 10 | 2 | écriture d'une requête d'interrogation sur une table | SELECT id\_match FROM joueur  JOIN equipe ON j\_1 = id\_joueur  JOIN match ON eq\_1 = id\_equipe  WHERE commune = 'Bois-Plage' ; |
| 11 | 2 | écriture d'une requête d'interrogation sur une table | SELECT DISTINCT joueur.nom, joueur.prenom  FROM joueur JOIN equipe ON j\_1 = id\_joueur  JOIN match ON eq\_1 = id\_equipe  WHERE eq\_gagnante = eq\_1  ORDER BY joueur.nom, joueur.prenom; |

# Exercice 2 (6 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | calculs en base 16 | Le code ASCII de 'E' est 0x45, celui de 'W' est 0x57. La clé sera donc 0x45 + 0x57 = 0x9C en hexadécimal ou 156 en décimal.  On ne demande pas la valeur décimale, on ignore une valeur décimale fausse. |
| 2 | 1 | raisonner sur une définition | Les deux mots étant des anagrammes, leur valeur de clé sera la même. |
| 3 | 1 | compléter une boucle simple | 2 somme = 0 3 for caractere in mot: 4 somme = somme + ord(caractere)  \* 0.25 pour initialiser la somme \* 0.25 pour le `for` \* 0.25 pour l'addition \*\*et\*\* l'utilisation de `ord`. |
| 4 | 1 | restituer des connaissances de cours | Permet de ne conserver que l’octet de poids faible pour la somme.  L’opérateur modulo (%) est utilisé, on ne conserve que le reste de la division entière par 0x100 = 256. La clé est donc un nombre entier qui tient sur un octet, elle peut prendre une valeur entre 0 et 255.  On peut répondre avec des valeurs hexadécimales (0xFF, ou 0x100 - 1). |
| 5 | 2 | calcul de complexité | On effectue une comparaison de mots par tour de boucle. Dans le pire cas, la boucle while tourne fois, donc on effectue comparaisons de mots. |
| 6 | 1 | connaissances de cours sur les dictionnaires | cle in dico |
| 7 | 3 | programmation | 1 def ajouter\_mot\_dict(dict\_mots, mot) : 2 cle = code\_hachage(mot) 3 if cle in dict\_mots : 4 ajouter\_mot\_liste(dict\_mots[cle], mot) 5 else: 6 dict\_mots[cle] = [mot] |
| 8 | 2 | c | * 1er appel : debut = 0 et fin = 5; * 2ème appel : debut = 0 et fin = 2; * 3ème et dernier appel : debut = 1 et fin = 1. |
| 9 | 2 | c | La méthode séquentielle est de complexité linéaire alors que la méthode dichotomique, qui divise par deux à chaque étape la sous liste dans laquelle la recherche s’effectue, est de complexité logarithmique. |
| 10 | 2 | connaissances de cours | La méthode effectue opérations.  On accepte toute réponse qui mentionne le coût logarithmique, même si la base est imprécise, y compris exprimé sous la forme “ tel que ”. |
| 11 | 2 | c | 1 def mot\_present(dictMots, mot) : 2 clé = code\_hachage(mot) 3 if clé in dictMots : 4 return est\_present(dictMots[clé], mot, 0, len(dictMots[clé])) 5 else : 6 return False |

# Exercice 3 (8 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | algorithmique | Trois chemins non prolongeables possibles :   * 3, 9, 1, 2, **4, 8** * 3, 9, 1, 2, **8, 4** * 3, 9, 1, 2, **6** |
| 2 | 1 | boucle for avec range | Les valeurs entières de 1 à n ou de 1 à 9. |
| 3 | 1 | attribut sur objet | 1 (car jeu\_9[0] représente le plus petit entier du jeu soit 1) |
| 4 | 2 | programmation objet | On teste ici deux conditions :   * que le sommet s pris dans le balayage de graphe est différent du sommet considéré, auquel cas il ne faut pas le considérer, sans quoi on introduirait une boucle (voire deux : une dans diviseurs et une dans multiples) ; * que la valeur du sommet s est un diviseur de celle du sommet considéré, auquel cas il faut l’ajouter à la liste des diviseurs et ajouter à la liste des multiples de s le sommet considéré. |
| 5 | 1 | liste/algo | Attention jeu\_9[5] représente le nombre 6, donc jeu\_9[5].diviseurs contiendra une liste d’objet de la classe Sommet dont les valeurs sont 1, 2 et 3 (les diviseurs de 6) |
| 6 | 2 | utilisation d'une méthode | def creer\_jeu(n):  """ renvoie le graphe du jeu à n sommets"""  jeu = []  for valeur in range(1, n+1) :  sommet = Sommet(valeur)  sommet.relier\_diviseurs(jeu)   jeu.append(sommet)  return graphe |
| 7 | 2 | liste par compréhension | def lister\_diviseurs(self):  return [sommet.valeur for sommet in self.diviseurs] |
| 8 | 2 | jeu de tests, diviseurs, multiples | l\_div\_3 = jeu[2].lister\_diviseurs()  l\_mult\_3 = jeu[2].lister\_multiples()  assert 1 in l\_div\_3  assert 6 in l\_mult\_3  assert 9 in l\_mult\_3 |
| 9 | 1 | assertion | Permet de provoquer une exception de type AssertionError si la méthode est exécutée sur une file vide, ce qui est justifié car dans ce cas il n’y a pas d’éléments à défiler. |
| 10 | 2 | gestion d'une file | def taille(self) :  return len(self.donnees) - self.decalage |
| 11 | 2 | gestion d'une file | |  |  | | --- | --- | | f = File() f.enfiler(1) f.enfiler(2) f.enfiler(3) assert (f.taille() == 3) assert (f.defiler() == 1) assert (f.defiler() == 2) assert (f.defiler() == 3) assert (f.est\_vide()) |  | |
| 12 | 3 | programmation | def rechercher\_chemins(jeu):  chemins\_np = []  f = File()  for sommet in jeu:  f.enfiler([sommet])  while not f.est\_vide():   chemin = f.defiler()   dernier = chemin[-1]   voisins = dernier.diviseurs + dernier.multiples  prolongeable = False   for voisin in voisins:  if voisin not in chemin:  prolongeable = True   f.enfiler(chemin + [voisin])   if not prolongeable:  chemins.append(chemin)  return chemins |
| 13 | 1 | boucle | def valeurs\_chemin(chemin):  return [ s.valeur for s in chemin ] |
| 14 | 2 | boucle | 1 chemins = []  2 for chemin in rechercher\_chemins(jeu\_9):  3 chemins.append( valeurs\_chemin(chemin) ) |
| 15 | 3 | recherche d'un maximum | def extraire\_plus\_longs\_chemins(chemins):  longueur\_max = 0  res = []  for chemin in chemins:  if len(chemin) > longueur\_max:  longueur\_max = len(chemin)  for chemin in chemins:  if len(chemin) == longueur\_max:  res.append(chemin)  return res |
| 16 | 3 | complexité | Des problèmes de temps d’exécution, de complexité, de profondeur de récursivité, qui peuvent apparaître avec un nombre de sommets très grand. |