# Exercice 1 (6 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | chaînes de caractères, algorithmique | 2 + 9 + 4 + 2 = 17 caractères, d’où 25 - 17 = 8 espaces |
| 2 | 1 | chaînes de caractères, algorithmique | Il y a 8 espaces au total à partager en 3 emplacements : 3 + 3 + 2. Seule la troisième proposition An---algorithm---must--be est correcte. |
| 3 | 1 | Programmation Python | assert nb\_caracteres + nb\_mots - 1 <= justification |
| 4 | 1 | Algorithmique, programmation Python | return liste\_mots[0] + " " \* nb\_espace\_total reponse = reponse + " " \* (q + 1) + liste\_mots[i] reponse = reponse + " " \* q + liste\_mots[i] |
| 5 | 2 | Algorithmique | Une façon de faire consisterait à placer *sur la première ligne le plus de mots possibles*, puis d’effectuer la même chose pour les lignes suivantes tant qu’il reste des mots à placer. |
| 6 | 2 | Algorithmique, programmation Python, structures de données | 9 for debut, fin in decoupage:  10 ligne\_justifiee = \ 11 ajout\_espace(liste\_mots[debut:fin], justification)  12 print(ligne\_justifiee)  ou  9 for couple in decoupage:  10 ligne\_justifiee = \ 11 ajout\_espace(liste\_mots[couple[0]:couple[1]], justification)  12 print(ligne\_justifiee) |
| 7 | 1 | Algorithmique | | Coût total du découpage : 147 | | | | | | | --- | --- | --- | --- | --- | --- | | Indice du mot de début | Indice du mot de fin + 1 | Nombre de mots | Nombre de caractères | Nombre d’espaces supplémentaires pour atteindre 15 caractères | coût | | 0 | 2 | 2 | 11 | 3 | 9 | | 2 | 4 | 2 | 6 | 8 | 64 | | 4 | 7 | 3 | 8 | 5 | 25 | | 7 | 8 | 1 | 8 | 7 | 49 | |
| 8 | 2 | Algorithmique, programmation Python | 1 def cout(i, j, liste\_mots, justification): 2 nb\_mots = j - i 3 nb\_caracteres = sum([len(liste\_mots[k]) for k in range(i,j)]) 4 if nb\_caracteres + (nb\_mots - 1) > justification: 5 return 1000000 6 else: 7 return (justification- nb\_caracteres - nb\_mots + 1) \*\* 2 |
| 9 | 1 | Algorithmique, Coût | “non ce n’est pas raisonnable” car croissance trop rapide.  référence à un coût de l’ordre de ou à une croissance en temps exponentielle. |
| 10 | 2 | Algorithmique, complexité | Coût quadratique, justification deux boucles, en au plus “n” étapes, imbriquées et/ou pour le calcul 1+2+3 + …+ n = (n\*(n+1))/2 |
| 11 | 3 | Algorithmique, programmation Python | cout\_mini[j] = mininmun(cout\_mini[j] + cout(i, j, liste\_mots, justification)) pour j compris entre i+1 et n exclu |
| 12 | 2 | Algorithmique, programmation Python | return [cout\_mini[0], decoupage] ou return (cout\_mini[0], decoupage) |

# Exercice 2 (6 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | Arbre binaire | La racine est le nœud contenant : ( -j-f-e-l-i-p-t-a-u, 19). Une feuille : (j, 1) (10 possibilités). |
| 2 | 1 | Arbre binaire | 4, 1100. |
| 3 | 3 | Analyser | Le codage de la phrase est plus compact car les caractères les plus fréquents ont des codes plus courts.  Notation : point accordé dès que l’idée est présente (peu profond = code court). |
| 4 | 2 | Programmation Orientée Objet | 1 class Noeud:  2 def \_\_init\_\_(self, nom, nb\_occu, fils\_g, fils\_d):  3 self.nom = nom   4 self.nb\_occu = nb\_occu   5 self.fils\_g = fils\_g   6 self.fils\_d = fils\_d   7   8 def \_\_str\_\_(self):  9 """ 10 Renvoie une chaine contenant les donnees 11 du noeud (nom et nombre d'occurrences) 12 """ 13 return '(' + self.nom + ', ' + str(self.nb\_occu) + ')' |
| 5 | 2 | Dictionnaire | 1 def liste\_occurrences(chaine):  2 dico = {}   3 for c in chaine:  4 if c in dico:   5 dico[c] = dico[c] + 1  6 else:  7 dico[c] = 1   8 liste\_res = []   9 for cle in dico: 10 liste\_res.append((cle, dico[cle]))  11 return liste\_res  dict() accepté en ligne 2 |
| 6 | 2 | Tri par insertion | 1 def tri\_liste(liste\_a\_trier):  2 liste\_triee = []  3 for i in range(0, len(liste\_a\_trier)):   4 element = liste\_a\_trier[i]  5 j = 0   6 while (j < len(liste\_triee) and  7 element[1] >= liste\_triee[j][1]):  8 j += 1   9 liste\_triee.insert(j, element)  10 return liste\_triee |
| 7 | 2 | Parcours séquentiel d'un tableau | 1 def conversion\_en\_noeuds(liste\_tuples): 2 liste\_noeuds = [] 3 for el in liste\_tuples: 4 liste\_noeuds.append(Noeud(el[0], el[1], None, None)) 5 return liste\_noeuds  ou retour de la liste créée directement par compréhension. |
| 8 | 2 | Parcours séquentiel d'un tableau | 1 def insere\_noeud(noeud, liste\_noeud): 2 j = 0 3 while j < len(liste\_noeud) and noeud.nb\_occu > liste\_noeud[j].nb\_occu:  4 j += 1  5 liste\_noeud.insert(j, noeud) |
| 9 | 3 | Traduire un algorithme en programme | 1 def construit\_arbre(liste):  2 while len(liste) > 1:   3 noeud1 = liste.pop(0)  4 noeud2 = liste.pop(0)  5 nom\_noeud\_pere = noeud1.nom + "-" + noeud2.nom  6 nb\_occu\_noeud\_pere = noeud1.nb\_occu + noeud2.nb\_occu   7 noeud\_pere = Noeud(nom\_noeud\_pere, nb\_occu\_noeud\_pere, noeud1, noeud2)   8 insere\_noeud(noeud\_pere, liste)   9 return liste[0] |
| 10 | 1 | Dictionnaire | Un dictionnaire. |
| 11 | 2 | Parcours séquentiel d'une chaîne de caractères | 1 def compresse(chaine, dico): 2 chaine\_resultat = "" 3 for c in chaine: 4 chaine\_resultat += dico[c] 5 return chaine\_resultat |

# Exercice 3 (8 points)

Correction

| Question | Niveau | Contenu | Solution |
| --- | --- | --- | --- |
| 1 | 1 | Accéder aux attributs d'une classe. | a4.duree = 12 |
| 2 | 1 | Structures linéaires de données | a2.voisines[2][1] vaut 4. En effet, a2.voisines est la liste des couples (voisine, distance) où voisine est une attraction voisine de a2 et distance est la distance (en minutes) entre a2 et voisine. Ainsi, a2.voisines[2] est le couple (a4,4). a2.voisines[2][1] correspond à l’élément d’indice 1 de (a4,4) soit 4, la durée de parcours entre a4 et a2. |
| 3 | 1 | Graphes : modéliser des situations sous forme de graphe. | Cette ligne affecte à a3.voisines une liste de 2-uplets. Chacun de ces 2-uplets est formé d’une voisine et de la distance (en minute) de cette attraction voisine à a3. Par exemple, (a1,5) signifie que a1 est une attraction voisine de a3 située à 5 minutes de a3. |
| 4 | 1 | Graphes : modéliser des situations sous forme de graphe. | 1 a4.voisines = [(a2,4),(a3,6)] |
| 5 | 2 | Graphes : modéliser des situations sous forme de graphe. | Cette modélisation du parc d’attractions est réalisée en utilisant un graphe non orienté car, lorsque deux attractions sont voisines, on peut aller de l’une à l’autre et de l’autre à l’une avec la même durée. |
| 6 | 2 | Analyser et modéliser un problème | Comme indiqué dans l’énoncé, il faut compter la durée de chaque attraction et la durée des trajets entre une attraction et la suivante dans la balade : 11+7+6+3+9 = 36 minutes. |
| 7 | 2 | Graphes : structures relationnelles | Le tableau [a2, a1, a4, a3] n’est pas une balade car dans le graphe du parc d’attraction, n’y a pas d’arêtes reliant a1 et a4. |
| 8 | 2 | Écrire une fonction simple en Python, à partir de rien. | 1 def sont\_voisines(attraction1, attraction2):  2 for couple in attraction1.voisines:  3 if couple[0] == attraction2:  4 return True  5 return False  6   7 def sont\_voisines2(attraction1, attraction2):  8 for attr, distance in attraction1.voisines:  9 if attr == attraction2: 10 return True 11 return False |
| 9 | 3 | Écrire une fonction simple en Python, à partir de rien | 1 def est\_balade(tab): 2 for i in range(len(tab)-1): 3 if not sont\_voisines(tab[i], tab[i+1]): 4 return False 5 return True  Il faut veiller à ce qu’il n’y ait pas d’accès illicite. |
| 10 | 1 | Algorithmes sur les graphes | Le parcours effectué est un parcours en profondeur. |
| 11 | 2 | Algorithmes sur les graphes | Le tableau obtenu est [a4, a2, a1, a3].  Si à la question précédente un parcours en largeur a été mentionné, un parcours compatible, ici [a4, a2, a3, a1] sera accepté.  Attention, si à la question 2 la réponse a4.voisines = [(a3,6),(a2,4)], le parcours en profondeur doit être [a4, a3, a1, a2] et le parcours en largeur doit être [a4, a3, a2, a1]. |
| 12 | 2 | Algorithmes sur les graphes | Le tableau obtenu est [a3, a1, a2, None]. |
| 13 | 2 | Distinguer les structures de données. | La variable deja\_vues est un dictionnaire qui prend comme clé le nom d’une attraction et qui lui associe le booléen vrai (True). Ce dictionnaire permet de savoir si un sommet du graphe a déjà été exploré lors du parcours. |
| 14 | 1 | Modèle relationnel | Une clé primaire d’une relation est l’attribut de cette relation qui permet d’identifier de manière unique un enregistrement. Une clé étrangère est une clé utilisée dans une relation pour faire référence à un enregistrement d’une autre table. |
| 15 | 1 | Langage SQL : requête simple | SELECT DISTINCT nom, prenom FROM visiteur WHERE date = '2025-01-11'; |
| 16 | 2 | Langage SQL : requête complexe | SELECT SUM(p.prix)  FROM visiteur JOIN photo  ON visiteur.id = photo.id\_visiteur WHERE visiteur.nom = "TURING"  AND prenom = "Alan"  AND date <= '2024-12-31'  AND date >= '2024-01-01';  Une solution utilisant LIKE '%2024' pourra être utilisée mais n’est pas exigible. Les solutions utilisant un alias avec AS seront acceptées également. |
| 17 | 2 | Langage SQL : requête complexe | Ils voulaient connaître les noms et prénoms des visiteurs faisant la Grande roue à 12h34 le 26 juillet 2024. |
| 18 | 3 | Modèle relationnel | On peut ajouter à cette base de données une nouvelle relation format pour stocker par format/support les informations suivantes : id, format, prix. Il suffit ensuite de retirer l’attribut prix de la relation photo et d’y ajouter une clé étrangère id\_format qui fait référence à la clé primaire id de la relation **format**. |