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.

<table>
<tr><th>Question</th><th>Niveau</th><th>Contenu</th><th>Solution</th></tr>
<tr><td>6</td><td>2</td><td>Algorithmique, programmation Python, structures de données</td><td>

```
 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)
```

</td></tr>
<tr><td>7</td><td>1</td><td>Algorithmique</td><td>Coût total du découpage : 147
<table>
<tr><th>Indice du mot de début</th><th>Indice du mot de fin + 1</th><th>Nombre de mots</th><th>Nombre de caractères</th><th>Nombre d’espaces supplémentaires pour atteindre 15 caractères</th><th>coût</th></tr>
<tr><td>0</td><td>2</td><td>2</td><td>11</td><td>3</td><td>9</td></tr>
<tr><td>2</td><td>4</td><td>2</td><td>6</td><td>8</td><td>64</td></tr>
<tr><td>4</td><td>7</td><td>3</td><td>8</td><td>5</td><td>25</td></tr>
</table>
</td></tr>
</table>

Question	*Niveau*	*Contenu*	*Solution*
			7 8 1 8 7 4 9

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 2^n 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, programm	`cout_mini[j] = mininmun(cout_mini[j] + cout(i, j, liste_mots, justification))` pour `j` compris entre `i+1` et `n` exclu

Question	*Niveau*	*Contenu*	*Solution*
		ation Python	
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	`class Noeud:`
			`    def __init__(self, nom, nb_occu, fils_g, fils_d):`
			`        self.nom = nom`
			`        self.nb_occu = nb_occu`
			`        self.fils_g = fils_g`
			`        self.fils_d = fils_d`
			
			`    def __str__(self):`
			`        """`
			`        Renvoie une chaine contenant les donnees`
			`        du noeud (nom et nombre d'occurrences)`
			`        """`

Question	*Niveau*	*Contenu*	*Solution*
			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`

Question	*Niveau*	*Contenu*	*Solution*
			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

Question	*Niveau*	*Contenu*	*Solution*
			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	`1 a4.voisines = [(a2,4),(a3,6)]`

Question	*Niveau*	*Contenu*	*Solution*
		des situations sous forme de graphe.	
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	`def sont_voisines(attraction1, attraction2):`
			`  for couple in attraction1.voisines:`
			`  if couple[0] == attraction2:`
			`  return True`

Question	*Niveau*	*Contenu*	*Solution*
		partir de rien.	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]`.

Question	*Niveau*	*Contenu*	*Solution*
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"`

Question	*Niveau*	*Contenu*	*Solution*

			`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**.