

Exercice 1 (6 points)

Correction

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|--------------------------|--|
| 1 | | 1 | Manipulation de tableaux | E P R U V D N S I A B C F G H J K L M O Q T X Y Z |
| 2 | | 1 | Manipulation de tableaux | assert 'W' not in clef |
| 3 | | 1 | Assertions | <pre>1 def creer_carre(liste_clef): 2 carre = [[0 for i in range(5)] for j in range(5)] 3 for i in range(25): 4 carre[i//5][i%5] = liste_clef[i] 5 return carre</pre> |
| 4 | | 1 | couper_en_digrammes | <pre>1 def couper_en_digrammes(message): 2 digrammes = [] 3 i = 0 4 while i < len(message) - 1: 5 if message[i] == message[i+1]: 6 digrammes.append(message[i] + 'X') 7 i = i + 1 8 else: 9 digrammes.append(message[i] +</pre> |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|--------------------------|--|
| | | | | <pre> message[i+1]) 10 i = i + 2 11 if i == len(message) - 1: <i>#il reste une lettre</i> <i>isolée</i> 12 digrammes.append(message[i] + 'X') 13 return digrammes </pre> |
| 5 | | 1 | Programmation | ["BO", "NJ", "OU", "RX"] |
| 6 | | 2 | Algorithme | BO -> HV NJ -> QG OU -> NV RX -> CU Soit le chiffré HVQGNVCU |
| 7 | | 1 | Manipulation de tableaux | <pre> 1 def ligne_colonne(lettre, carre): 2 for i in range(len(carre)): 3 for j in range(len(carre)): 4 if carre[i][j] == lettre: 5 return (i,j) </pre> |
| 8 | | 1 | Manipulation de tableaux | <pre> 1 def sur_la_meme_ligne(digramme, carre): 2 lettre1 = digramme[0] 3 lettre2 = digramme[1] </pre> |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|---------------|---|
| | | | | <pre> 4 return ligne_colonne(lettre1, carre)[0]==ligne_colonne(lettre2, carre)[0] </pre> |
| 9 | | 3 | Programmation | <pre> 1 def chiffrer_digramme(digramme, carre): 2 lettre1 = digramme[0] 3 lettre2 = digramme[1] 4 i1, j1 = ligne_colonne(lettre1, carre) 5 i2, j2 = ligne_colonne(lettre2, carre) 6 if sur_la_meme_ligne(digramme, carre): 7 digramme_chiffre = carre[i1][(j1 + 1)%5] + carre[i2][(j2 + 1)%5] 8 elif sur_la_meme_colonne(digramme, carre): 9 digramme_chiffre = carre[(i1 + 1)%5][j1] + carre[(i2 + 1)%5][j2] 10 else: 11 digramme_chiffre = carre[i1][j2] + carre[i2][j1] 12 return digramme_chiffre </pre> |
| 10 | | 3 | Programmation | <pre> 1 def chiffrer_playfair(message, clef): 2 carre = creer_carre(creer_liste_clef(clef)) 3 d_msg = couper_en_digrammes(message) 4 msg_chiffre = "" 5 for d in d_msg: 6 msg_chiffre += chiffrer_digramme(d, carre) 7 return msg_chiffre </pre> |

Exercice 2 (6 points)

Correction

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|--|--|
| 1 | | 1 | Utiliser les commandes de base en ligne de commande | La commande <code>ls</code> permet d'obtenir le listing des fichiers et dossiers présents. On ne demande pas l'option <code>-l</code> . |
| 2 | | 1 | Gérer les droits et permissions d'accès aux fichiers | <code>-rwxrw-r--</code> signifie que le propriétaire peut lire, écrire/modifier et exécuter le fichier. les autres membres du groupe peuvent lire et écrire/modifier les autres utilisateurs ne peuvent que le lire. |
| 3 | | 1 | Utiliser les commandes de base en ligne de commande | <code>rm concat.pls</code> . On accepte <code>rm</code> seul. |
| 4 | | 1 | Utiliser la documentation d'une bibliothèque | <code>chmod g-wx</code> indique que les permissions en écriture et en exécution sont retirées aux membres du groupe <code>admins</code> . |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|--|---|
| | | | e, d'une commande. | |
| 5 | | 1 | Passer de la représentation d'une base dans une autre. | rw- donne 110 qui vaut 6 |
| 6 | | 2 | Programmation algorithmique | <pre> 1 def bin_to_oct(chaine): 2 valeur = 0 3 for i in range(len(chaine)): 4 valeur = valeur + int(chaine[i]) * 2 ** (2 - i) 5 return valeur ou 1 def bin_to_oct(chaine): 2 valeur = 0 3 poids = 1 4 for i in range(len(chaine)-1, -1, -1): 5 valeur = valeur + int(chaine[i]) * poids 6 poids = poids * 2 7 return valeur </pre> <p>ou tout autre réponse valide.</p> |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|---|--|
| 7 | | 2 | Algorithmique | On obtient en sortie la chaîne '110101100'. |
| 8 | | 2 | Programmation algorithmique | <pre> 1 def symb_to_oct(chaine): 2 repr_bin = symb_to_bin(chaine) 3 # les 3 premiers bits correspondent au propriétaire 4 user = repr_bin[0] + repr_bin[1] + repr_bin[2] 5 # les 3 suivants au groupe 6 group = repr_bin[3] + repr_bin[4] + repr_bin[5] 7 # et les 3 derniers aux autres 8 other = repr_bin[6] + repr_bin[7] + repr_bin[8] 9 return str(bin_to_oct(user)) + str(bin_to_oct(group)) + str(bin_to_oct(other)) </pre> ou toute autre réponse valide en une ou plusieurs lignes. |
| 9 | | 1 | Vocabulaire et utilisation de la programmation objet. | <pre> mon_fichier = Fichier('concat.pls', 1257, 'root', 'www- data', 'rw-r--r--') </pre> |
| 10 | | 2 | Programmation objet | <p>on attend une méthode</p> <pre> 1 def chown(self, nouveau_proprietaire): 2 self.proprietaire = nouveau_proprietaire </pre> |
| 11 | | 2 | Programmation objet | <pre> 1 def get_executable(fichiers, proprietaire): 2 executables = [] </pre> |

| <i>Question</i> | <i>Barème</i> | <i>Niveau</i> | <i>Contenu</i> | <i>Solution</i> |
|-----------------|---------------|---------------|----------------|--|
| | | | | <pre>3 for fichier in fichiers: 4 if fichier.propretaire == proprietaire and fichier.permission[2] == 'x': 5 executables.append(fichier) 6 return executables</pre> |

Exercice 3 (8 points)

Correction

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|---|---|
| 1 | | 1 | BDD insertion | <code>INSERT INTO Produits VALUES (10,'Croissant', 'Alimentaire', 0.90, 4)</code> |
| 2 | | 1 | BDD mise à jour | <code>UPDATE Fournisseurs SET adresse = '78 Rue des Jeux', ville = 'Elbeuf', WHERE nom_fournisseur = 'Livres en Folie';</code> |
| 3 | | 1 | BDD comprendre requête SELECT FROM WHERE | <p>Cette requête SQL sélectionne et affiche les noms de tous les produits de la catégorie Alimentaire.</p> <p>nom</p> <hr/> <p>Yaourts blanc x 4</p> <p>Lait</p> <p>Pain</p> <p>On accepte la réponse sous la forme d'une phrase ou d'un tableau.</p> |
| 4 | | 1 | BDD requête avec 1 condition | <code>SELECT * FROM Commandes WHERE total >= 1000;</code> |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|---|--|
| 5 | | 1 | BDD requête condition double OR | <pre>SELECT nom FROM Fournisseurs WHERE pays = 'Espagne' OR pays = 'Italie' ;</pre> |
| 6 | | 2 | BDD requête jointure | <pre>SELECT Fournisseurs.nom FROM Fournisseurs JOIN Produits ON Fournisseurs.id_fournisseur = Produits.id_fournisseur WHERE catégorie = 'Alimentaire';</pre> |
| 7 | | 3 | requête 3 jointures | <pre>SELECT id_commande, date, Fournisseurs.nom FROM Fournisseurs JOIN Commandes ON Fournisseurs.id_fournisseur = Commandes.id_fournisseur JOIN Details ON Commandes.id_commande = Details.id_commande JOIN Produits ON Details.id_produit = Produits.id_produit WHERE categorie = 'Vêtement';</pre> |
| 8 | | 1 | Graphe chemin & algo de Dijkstra | Le plus court chemin est : Toulouse - Bordeaux - Nantes - Calais |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|---|--|
| 9 | | 2 | Graphe Parcours profondeur | Calais – Nantes – Bordeaux – Toulouse – Lyon – Marseille - Paris – Strasbourg |
| 10 | | 2 | Graphe Parcours profondeur | Calais – Nantes – Paris – Strasbourg – Bordeaux – Lyon – Toulouse – Marseille |
| 11 | | 1 | ajout dans un dictionnair e | <pre> 1 graphe['Lyon']['Marseille'] = 315 2 graphe['Marseille']['Lyon'] = 315 3 </pre> |
| 12 | | 2 | parcours dico | <pre> 1 def distance(graphe, ville1, ville2): 2 if ville1 in graphe and ville2 in graphe[ville1]: 3 return graphe[ville1][ville2] 4 else: 5 return None </pre> |
| 13 | | 3 | calcul d'une somme avec parcours d'un cido | <pre> 1 def distance_totale(graphe, ville_depart, ville_destination): 2 chemin = trouver_chemin(graphe, ville_depart, ville_destination) 3 if chemin is None: 4 return None 5 6 distance_totale = 0 7 for i in range(len(chemin) - 1): 8 ville_courante = chemin[i] </pre> |

| Question | Barème | Niveau | Contenu | Solution |
|----------|--------|--------|--|---|
| | | | | <pre> 9 prochaine_ville = chemin[i + 1] 10 distance_totale += graphe[ville_courante][prochaine_ville] 11 return distance_totale </pre> |
| 14 | | 1 | accès à une info dans une liste qui est une valeur d'un dico | La valeur de <code>graphe['Paris']['Nantes'][1]</code> est 3.47. |
| 15 | | 2 | compléter code - utilisation dico | <pre> 1 def ratio_duree_distance(graphe): 2 for ville, connexions in graphe.items(): 3 for destination, valeurs in connexions.items(): 4 distance, duree = valeurs 5 ratio = duree / distance 6 graphe[ville][destination].append(ratio) 7 return graphe </pre> |
| 16 | | 1 | comprendre un algo -> algo glouton | Algorithme glouton. |
| 17 | | 2 | algo glouton | <code>['Toulouse', 'Bordeaux', 'Nantes', 'Paris', 'Calais']</code> |