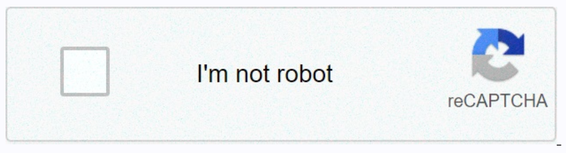
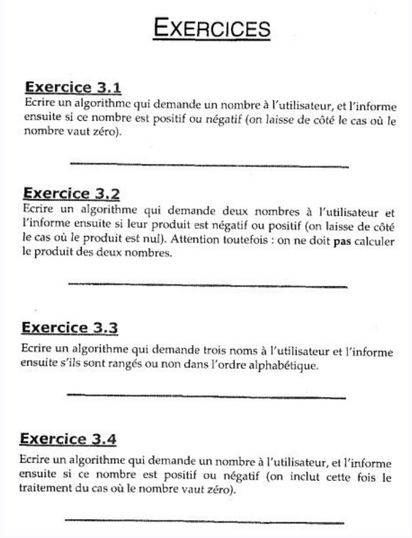


Continue

Exercice corrigé algorithme chaine de caractère pdf

Exercices d'algorithmique ❧ Calculer le nombre d'occurrences d'un caractère : def compte_car(s: str, c: str) -> int: """ :entrée s: str :entrée c: str :pré-cond: len(c) == 1 :sortie n: int :post-cond: n est le nombre de valeurs i telles que s[i] == c """ Exemple : compte_car("hello", "l") # retourne 2 compte_car("hello", "z") # retourne 0 compte_car("hello", "H") # retourne 0 (différence entre "h" et "H") ❧ Calculer le nombre de lettres minuscules : def compte_minuscules(s: str) -> int: """ :entrée s: str :sortie n: int :post-cond: n est le nombre valeurs de i telles que s[i] est une minuscule. """ Pour cet exercice, on se limitera aux 26 lettres simples de l'alphabet, sans considérer les caractères accentués ou altérés. ❧ Déterminer l'indice minimum (le plus à gauche) d'un caractère : def indice_min_car(s: str, c: str) -> int: """ :entrée s: str :entrée c: str :pré-cond: len(c) == 1 :sortie imin: int ou None :post-cond: imin est la plus petite valeur telle que s[imin] == c , ou None si s ne contient pas c. """ ❧ Déterminer l'indice maximum (le plus à droite) d'un caractère : def indice_max_car(s: str, c: str) -> int: """ :entrée s: str :entrée c: str :pré-cond: len(c) == 1 :sortie imax: int ou None :post-cond: imax est la plus grande valeur telle que s[imax] == c , ou None si s ne contient pas c. """ ❧ Déterminer l'indice suivant (c.à.d. plus à droite) d'un caractère : def indice_suivant_car(s: str, c: str, ig: int) -> int: """ :entrée s: str :entrée c: str :entrée ig: int :pré-cond: len(c) == 1 :sortie ic: int ou None :post-cond: ic est la plus petite valeur telle que ig < ic et s[ic] == c , ou None si s[ig+1:] ne contient pas c. """ ❧ Déterminer l'indice précédent (c.à.d. plus à gauche) d'un caractère : def indice_prec_car(s: str, c: str, id: int) -> int: """ :entrée s: str :entrée c: str :entrée id: int :pré-cond: len(c) == 1 :sortie ic: int ou None :post-cond: ic est la plus grande valeur telle que ic < id et s[ic] == c , ou None si s[:id] ne contient pas c.
""" ❧ Déterminer si une chaîne commence par une autre : def commence_par(s: str, t: str) -> bool: """ :entrée s: str :entrée t: str :sortie c: bool :post-cond: c est True si et seulement si pour tout i tel que 0 ≤ i < len(t) , s[i] == t[i] . """ ❧ Déterminer l'indice minimum (le plus à gauche) d'une sous-chaîne : def indice_min(s: str, t: str) -> int: """ :entrée s: str :entrée t: str :sortie imin: int ou None :post-cond: imin est la plus petite valeur telle que, pour tout i tel que 0 ≤ i < len(t), s[imin+i] == t[i] , ou None si s n'admet pas t pour sous-chaîne. """ ❧ Déterminer l'indice maximum (le plus à gauche) d'une sous-chaîne : def indice_max(s: str, t: str) -> int: """ :entrée s: str :entrée t: str :sortie imax: int ou None :post-cond: imax est la plus grande valeur telle que, pour tout i tel que 0 ≤ i < len(t), s[imax+i] == t[i] , ou None si s n'admet pas t pour sous-chaîne. """ ❧ Déterminer l'indice suivant (c.à.d. plus à droite) d'une sous-chaîne : def indice_suivant(s: str, t: str, ig: int) -> int: """ :entrée s: str :entrée t: str :entrée ig: int :pré-cond: len(c) == 1 :sortie it: int ou None :post-cond: it est la plus petite valeur telle que ig < it et pour tout i tel que 0 ≤ i < len(t), s[imin+i] == t[i] , ou None si s[ig+1:] n'admet pas t pour sous-chaîne. """ ❧ Déterminer l'indice précédent (c.à.d. plus à gauche) d'une sous-chaîne : def indice_prec(s: str, t: str, id: int) -> int: """ :entrée s: str :entrée t: str :entrée id: int :sortie it: int ou None :pré-cond: len(c) == 1 :post-cond: it est la plus petite valeur telle que it < id et pour tout i tel que 0 ≤ i < len(t), s[imin+i] == t[i] , ou None si s[:id] n'admet pas t pour sous-chaîne. """ ❧ Calcule une chaîne inversée : def inverse(s: str) -> str: """ :entrée s: str :sortie t: str :post-cond: len(t) == len(s) et pour tout i tel que 0 ≤ i < len(s), t[i] == s[-i-1] . """ ❧ Détermine si une chaîne est un palindrome : def palindrome(s: str) -> bool: """ :entrée s: str :sortie p: bool :post-cond: p est True si et seulement si pour tout i tel que 0 ≤ i < len(s), s[i] == s[-i-1] . """ ❧ Compte le nombre de mots dans une chaîne : def compte_mots(s: str) -> int: """ :entrée s: str :sortie m: int :post-cond: m est le nombre de mots dans s """ On considère comme un mot toute séquence de caractères différents de l'espace (même si ce ne sont pas des lettres : chiffres, symboles de ponctuation…). Compter les mots consiste donc à compter le nombre de « non-espaces » situés juste après une espace (ou en début de chaîne). ❧ Vérifier si une chaîne de caractères est bien parenthésées : def bien_parenthesee(txt: str) -> bool: """ :entrée txt: str :sortie bp: bool :post-cond: bp est True si et seulement si txt est bien parenthésée """ Toute parenthèse ouverte doit ensuite être fermée, et une parenthèse ne peut pas être fermée si elle n'a pas été préalablement ouverte1. Le tableau ci-dessous donne des exemples de chaînes bien et mal parenthésées. Bien parenthésées Mal parenthésées abc ((abc)) ab(cd)ef abc) a(b)c(d)e ab)c a((b)c)d a(b(c)d a(b(c()e)f)g a(b)c)d(e)f ❧ Calculer la valeur numérique d'un entier représenté en binaire par une chaîne de caractères : def eval_binaire(txt: str) -> int: """ :entrée txt: str :pré-cond: txt ne contient que des caractères de ``0`` et ``1`` :sortie val: int :post-cond: val est la valeur de l'entier représenté (en base 2) par txt """ Vous n'utiliserez bien sûr pas la fonction int de Python, qui permet de faire cela. ❧ Calculer la valeur numérique d'un entier représenté par une chaîne de caractères : def eval_decimal(txt): """ :entrée txt: str :pré-cond: txt ne contient que des caractères de ``0`` à ``9`` :sortie val: int :post-cond: val est la valeur de l'entier représenté (en base 10) par txt """ Vous n'utiliserez bien sûr pas la fonction int de Python, qui permet de faire cela. NB: bien que ce ne soit pas obligatoire, l'algorithme peut-être simplifié en utilisant la fonction ord(c), qui retourne le code numérique (un entier) du caractère c, et en exploitant le fait que les codes des caractères numériques se suivent, donc ord('1') == ord('0')+1, ord('2') == ord('1')+1, etc. Variante : on autorise maintenant le premier caractère à être le signe moins -. ❧ Calculer la représentation binaire d'un entier : def repr_binaire(val: int) -> str: """ :entrée val: int :pré-cond: val >= 0 :sortie txt: str :post-cond: txt est la représentation binaire de val """ Vous n'utiliserez bien sûr pas la fonction format de Python, qui permet de faire cela. Variante : on autorise maintenant val à être négatif. ❧ Calculer la représentation en base 10 d'un entier : def repr_decimal(val: int) -> str: """ :entrée val: int :pré-cond: val >= 0 :sortie txt: str :post-cond: txt est la représentation en base 10 de val """ Vous n'utiliserez bien sûr pas les fonctions format ou str de Python, qui permettent de faire cela.



EXERCICES

Exercice 3.1

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on laisse de côté le cas où le nombre vaut zéro).

Exercice 3.2

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si leur produit est négatif ou positif (on laisse de côté le cas où le produit est nul). Attention toutefois : on ne doit **pas** calculer le produit des deux nombres.

Exercice 3.3

Ecrire un algorithme qui demande trois noms à l'utilisateur et l'informe ensuite s'ils sont rangés ou non dans l'ordre alphabétique.

Exercice 3.4

Ecrire un algorithme qui demande un nombre à l'utilisateur, et l'informe ensuite si ce nombre est positif ou négatif (on inclut cette fois le traitement du cas où le nombre vaut zéro).

NB: bien que ce ne soit pas obligatoire, l'algorithme peut-être simplifié en utilisant la fonction ord(c), qui retourne le code numérique (un entier) du caractère c, la fonction chr(i), qui retourne le caractère ayant \(i\) pour code numérique, et le fait que les codes des caractères numériques se suivent, donc chr(ord('0') + 1) == '1', chr(ord('0') + 2) == '2', etc. Variante : on autorise maintenant val à être négatif. ❧ Calculer la représentation hexadécimale (en base 16) d'un entier : def repr_hexadecimal(val: int) -> str: """ :entrée val: int :sortie txt: str :pré-cond: val >= 0 :post-cond: txt est la représentation en base 10 de val """ On rappelle qu'en hexadécimal, on utilise 16 chiffres, de 0 à 9 et de A à F. Vous n'utiliserez bien sûr pas la fonction format de Python, qui permet de faire cela. NB: On pourra utiliser, comme dans l'exercice précédent, les fonctions ord et chr, mais en faisant attention au fait que chr(ord('0') + 10) n'est pas égal à 'A'… ❧ Décompresse une chaîne de caractère : def decompresse(comp: str) -> str: """ :entrée comp: str :pré-cond: len(comp) est paire; tous les caractères d'indice pair sont des chiffres (entre 0 et 9) :sortie decomp: str :post-cond: 'décomp' est calculé en répétant chaque caractère d'indice impair de 'comp' par la valeur qui le précède par exemple "3a0b1c2a49" -> "aaacaa9999" """ ❧ Compresse une chaîne de caractère : def compresse(txt: str) -> str: """ :entrée txt: str :pré-cond: len(str) est paire; tous les caractères d'indice pair sont des chiffres (entre 0 et 9) :sortie compresse: str :post-cond: 'compressé' est la une chaîne telle que ``decompresse(compresse)`` donne ``txt`` (cf.
l'algo ``decompresse`` ci-dessus) """ Indices 1 Une solution consiste donc à parcourir la chaîne de gauche à droite en maintenant un compte du nombre de parenthèses ouvertes et non encore fermées.