Première NSI - Algorithmie

Travaux dirigés : parcours séquentiel

qkzk

2022/07/06

Se tester

1. Appartenance

1. Quel est le rôle du mot clé `in` en Python ? Citez deux usages courants.
2. Que retournent ces instructions ?

```
>>> 3 in [1, 2, 7]
>>> 4 in range(12)
>>> '4' in range(12)
>>> "ZA" in "AZALEE"
```

3. On rappelle cet algorithme, vu en cours :

```
fonction (tableau T, objet x) ---> booléen:
  Pour chaque élément e de T,
    Si e = x, alors on retourne Vrai
  Si la boucle se termine, on retourne Faux.
```

   a. Quel est le rôle de cet algorithme ?
   b. Écrire un programme Python qui transcrive cet algorithme.
   c. Parmi les exemples présentés à la question 2, quels sont ceux pour lesquels votre programme Python obtient la même réponse que l'instruction avec `in` ?

4. Adapter l'algorithme précédent pour qu'il retourne soit :

   - le premier indice de x si x figure dans le tableau,
   - -1 si x ne figure pas dans le tableau.

2. Recherche d'un élément maximal

On rappelle l'algorithme vu en cours :

```
fonction maximum(tableau T, nombre x) ---> nombre:
  On affecte à max la valeur de l'élément d'indice 0 du tableau.
  Pour chaque élément e du tableau:
    si e > max:
      max = e
  retourner max
```

1. Traduire cet algorithme en Python.
2. Quel est la complexité de cet algorithme ?
3. Démontrer que l'algorithme se termine.
4. Déterminer un invariant pour la boucle de l'algorithme.

S'entraîner

3. Renverser un tableau

1. Proposer un algorithme qui prend un tableau et le retourne. Voici la signature attendue :

   `retourner(tableau T) ---> tableau:`

   Ainsi qu'un exemple :

   ```
   >>> retourner([1, 2, 3])
   [3, 2, 1]
   ```

2. Traduire cet algorithme en Python.
   On pourra (mais ce n'est pas la seule approche) utiliser l'opérateur `+` pour les objets `list` qui *concatène* deux listes :

   ```
   >>> [1, 2] + [3, 4]
   [1, 2, 3, 4]
   ```

4. Décompte d'éléments

1. Proposer un algorithme qui compte le nombre d'apparitions d'un élément dans un tableau.

   `decompte(tableau T, element x) ---> int`

   Exemple :

   ```
   >>> decompte([1, 1, 2, 1, 1, 2, 1], 1)
   5
   >>> decompte(['a', 'b', 'ab', 'a'], 'a')
   2
   ```

2. Traduire cet algorithme en Python.
3. Adapter votre fonction en une fonction `frequence` qui calcule la fréquence d'un élément dans un tableau. (fréquence = effectif de la valeur divisé par effectif total)

5. Fonction mystere

Voici une fonction mystère :

```python
def mystere(mot: str) -> bool:
    i = 0
    j = len(mot) - 1
    while i <= j:
        if mot[i] != mot[j]:
            return False
        i = i + 1
        j = j - 1
    return True
```

1. Faire tourner cette fonction sur les mots `radar` et `radrar` (qui n'existe pas, je sais).
2. Quel est le rôle de cet algorithme ?
3. Démontrer que l'algorithme se termine.
4. Adapter le programme afin qu'il ne compte plus qu'une seule instruction `return`.
5. Proposer un algorithme alternatif réalisant la même chose utilisant une fonction introduite dans un exercice précédent.

6. `positions`

La fonction `positions` reçoit un tableau `t` et un objet `x` et renvoie la liste des indices de `x` dans `t`. Malheureusement ses instructions sont dans le désordre et l'indentation est fausse. Rectifier.

```
pos.append(i)
for i in range(len(t)):
if e == x:
e = t[i]
return pos
def positions(t: list, x) -> list:
pos = []
```

7. Séparer

Écrire une fonction `separer` permettant, à partir d'une liste de nombres, d'obtenir deux listes. La première comporte les nombres inférieurs ou égaux à un nombre donné, la seconde les nombres qui lui sont strictement supérieurs.

```
>>> separer([45, 21, 56 ,12, 1, 8, 30, 22, 6, 33], 30)
([21, 12, 1, 8, 30, 22, 6], [45, 56, 33])
```

8. Correction de l'algorithme `factorielle`

On rappelle la définition de la factorielle :

$0! = 1$ et $n! = 1 \times 2 \times 3 \times \cdots \times n$

Ainsi $1! = 1,\ 2! = 2,\ 3! = 6,\ 4! = 24$ etc.

On considère l'algorithme suivant :

```
factorielle(entier n) ---> entier:
    i = 0
    f = 1
    Tant que i < n:
        i = i + 1
        f = f * i
    renvoyer f
```

1. Vérifier les exemples cités plus haut pour cette fonction.
2. Justifier que la boucle n'est pas infinie.
3. Montrer, en utilisant la pré-condition $n \geq 0$, que la propriété :

$$f = i! \text{ et } i \geq n$$

   est un invariant de la boucle.

4. Formuler une post-condition qui établit la correction de cette algorithme.

9. Puissance de 2

1. Écrire un algorithme utilisant une boucle *tant que* permettant de déterminer si un entier n strictement positif est une puissance de 2.
2. Montrer que la boucle *tant que* se termine.
3. Montrer que l'algorithme produit le résultat attendu.

10. Le plus vieux

On dispose d'une liste de personnes et de leurs ages sous cette forme :

```
personnes = [('Joe', 16), ('Zoé', 17), ('Martin', 15)]
```

1. Proposer un algorithme qui détermine le plus âgé des individus.

   ```
   >>> plus_ages(personnes)
   'Zoé'
   ```

2. Déterminer un invariant pour la boucle de l'algorithme.
3. Traduire votre algorithme en Python.

Autres algorithmes

11. Meilleur ami

On a demandé aux enfants d'une classe le nom de leur meilleur ami. On a enregistré tous les noms dans un dictionnaire :

Par exemple :

```
exemple_1 = {
    "Pierre": "Paul",
    "Fanny": "Léa",
    "Chloe": "Sadia",
    "Paul": "Léna",
    "Léa": "Sadia",
    "Léna": "Chloé",
    "Sadia": "Léna",
}
```

Tous les enfants ont été interrogés et ils ont tous donné le prénom d'un élève de la classe. On suppose que tous les enfants ont un prénom différent.

On voudrait s'assurer que chaque enfant est *le meilleur ami* d'un autre et qu'aucun n'est isolé dans la classe.

Partons d'une situation simple :

```
exemple_2 = {
    "Pierre": "Paul",
    "Paul": "Pierre",
    "Jacques": "Pierre"
}
```

Dans cet exemple, Jacques n'est le meilleur ami de personne...

On décide de tester ainsi :

1. On crée une liste des amis `l`.
2. On part d'un prénom au hasard `p`.
3. On consulte le meilleur ami de `p` noté `a`.
4. Tant que `a` n'est pas déja dans `l`, on l'ajoute à `l` et `p = a`
5. On recommence à l'étape 3.

Questions

1. Cet algorithme s'arrête-il toujours ?
2. Que produit-il comme liste `l` pour l'exemple 2 si on a choisi de commencer par "Pierre" ? Par "Jacques" ?
3. Comment peut-on savoir que "Jacques" n'est le meilleur ami de personne ?

4. Faire tourner l'algorithme pour l'exemple 1 en partant de Pierre. Relever les `amis`.
5. Quels enfants ne sont pas cités à cette étape ?
6. Choisir un enfant non cité et recommencer *en maintenant la liste des amis produite à l'étape précédente.*
7. S'il reste des enfants non cités, recommencer en partant d'eux.
8. En appliquant cet algorithme assez de fois, on peut être certain de connaître tous les "meilleurs amis". Comparer la liste des amis à celle des élèves, on devrait pouvoir repérer ceux qui ne sont le meilleur ami de personne `:_(`

Beaucoup plus efficace avec les outils natifs de python.

```
eleves = set(exemple_1.keys())
amis = set(exemple_2.values())
isoles = eleves.difference(amis)
```

Un `set` (*ensemble*) est une structure qui permet d'enregistrer des éléments sans ordre particulier et sans doublon.

```
>>> s = set()
>>> s.add("Jean")
>>> s
{'Jean'}
>>> s.add("Pierre")
>>> s
{'Pierre', 'Jean'}
>>> s.add("Jean")
>>> s
{'Pierre', 'Jean'}
```

La méthode `.difference` renvoie les éléments du premier `set` qui ne sont pas dans le second `set`.

Ainsi, on obtient immédiatement les noms des enfants non cités.

12. Est inclus

L'ADN peut être représenté par une chaine de caractères formée avec les lettres `A`, `T`, `G`, `C`.

- Un **brin** est un petit morceau d'ADN, que l'on retrouve parfois dans
- un **gène** qui est une grande séquence d'ADN.

La fonction `est_inclus` prend en paramètres deux chaines de caractères `brin` et `gene` et renvoie la réponse, un booléen, à la question « Retrouve-t-on `brin` inclus dans `gene` ? ».

Cette fonction utilise une fonction auxiliaire : `correspond(motif, chaine, position)` qui renvoie `True` si on retrouve `motif` exactement à partir de `position` dans `chaine` et `False` sinon.

Compléter le code Python ci-dessous.

```
def correspond(motif, chaine, position):
    if ... > len(chaine):
        return False
    for i in range(len(motif)):
        if chaine[position + ...] != ...:
            return ...
    return True

def est_inclus(brin, gene):
    taille_gene = len(gene)
    taille_brin = len(brin)
    for i in range(... - ... + 1):
        if correspond(..., ..., ...):
            return True
    return False
```

```
# tests
assert correspond("AA", "AAGGTTCC", 4) == False
assert correspond("AT", "ATGCATGC", 4) == True

assert est_inclus("AATC", "GTACAAATCTTGCC") == True
assert est_inclus("AGTC", "GTACAAATCTTGCC") == False
assert est_inclus("AGTC", "GTACAAATCTTGCA") == False
assert est_inclus("AGTC", "GTACAAATCTAGTC") == True
```

13. Tester si un nombre est palindromique

Un mot palindrome peut se lire de la même façon de gauche à droite ou de droite à gauche : `'esse'`, `'radar'`, et `'non'` sont des mots palindromes.

De même certains nombres sont eux-aussi palindromiques : 33, 121, 345543.

L'objectif de cet exercice est d'obtenir un programme Python permettant de tester si un nombre est palindromique, quand sa représentation décimale est un palindrome.

Pour remplir cette tâche, on vous demande de compléter le code des trois fonctions ci-dessous sachant que la fonction `est_palindromique` s'appuiera sur la fonction `est_palindrome` qui elle-même s'appuiera sur la fonction `inverse_chaine`.

La fonction `inverse_chaine` inverse l'ordre des caractères d'une chaine de caractères `chaine` et renvoie la chaine inversée.

La fonction `est_palindrome` teste si une chaine de caractères `chaine` est un palindrome. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

La fonction `est_palindromique` teste si un nombre `nombre` est palindromique. Elle renvoie `True` si c'est le cas et `False` sinon. Cette fonction s'appuie sur la fonction précédente.

Compléter le code des trois fonctions ci-dessous.

```
def inverse_chaine(chaine):
    resultat = ...
    for caractere in chaine:
        resultat = ...
    return resultat

def est_palindrome(chaine):
    inverse = inverse_chaine(chaine)
    return ...

def est_palindromique(nombre):
    chaine = ...
    return est_palindrome(chaine)
```

```
# tests
assert inverse_chaine('bac') == 'cab'
assert not est_palindrome('NSI')
assert est_palindrome('ISN-NSI')
assert not est_palindromique(214312)
assert est_palindromique(213312)
```