Quizz_2020_2

January 26, 2020

0.1 Quizz 22 janvier 2020

Ecrire une *fonction* python `palindrome(c)` qui, étant donnée une variable c de type str (chaîne de caractères), renvoie True si la chaîne c est un palindrome, et False sinon.

Seules fonctions prédéfinies utilisables : `len()` et `range()`.

Cette version est la **deuxième version** du corrigé, car il y avait une erreur dans la première version: j'avais utilisé len(c)%2 (reste de la division entière de len(c) par 2) au lieu d'utiliser, conformément au commentaire de la première fonction, "len(c)//2", c'est-à-dire le quotient de la division entière de len(c) par 2, c'est-à-dire la moitié de la chaîne (+ ou -1 selon que la chaîne est de longueur paire ou impaire).

L'erreur se trouvait dans la fonction "palindrome()" et dans la fonction "palindromeC".

Mon jeu d'essai ne permettait pas de trouver facilement l'erreur: en effet, mes exemples de non palindromes avaient la propriété d'avoir la première lettre et la dernière lettre différente. Donc même avec cette erreur les fonctions palindrome() et palindromeC() répondait bien False pour ces non-palindromes. Mais si on ajoute un non-palindrome qui n'a pas cette propriété, comme "abtotoba", alors mes fonctions originales ne fonctionnaient pas (je vous invite à vérifier).

J'ai aussi ajouté une variante (palindromeD()) qui commence par inverser la chaîne initiale, et fait ensuite une comparaison de l'original et de la chaîne inversée. Cette variante, inspirée de vos réponses, fonctionne mais est plus coûteuse pour deux raisons : d'une part il y a deux boucles (la boucle d'inversion et la boucle cachée sous la comparaison de chaînes) et d'autre part la première boucle est plus longue puisqu'elle va jusqu'au bout de la chaîne dans tous les cas.

[1]:
```
def palindrome(c):
    for i in range(len(c)//2):
        if c[i] != c[-i-1]:
            return False
    return True
# pour i allant de 1 à la moitié de len(c)
# on compare le i-ième caractère au (len(c)-i)-ième
# (on pourrait aussi écrire c[len(c)-i-1])
# à la première différence on interromp la boucle et on renvoie False
# si la boucle a été jusqu'au bout: aucune différence trouvée,
# on renvoie True
```

Ci-dessous, d'autres solutions sont présentées, qui mettent en oeuvre différentes stratégies

[2]:
```python
# Un variante avec deux indices gérés "à la main": un pour les caractères
# en partant du début (d), et un pour les caractères en partant de la fin.
# Comme précédemment, la fonction s'arrête dès qu'on trouve une différence
def palindromeA(c):
    d = 0
    f = len(c)
    while d < f:
        if c[d] != c[f-1]:
            return False
        d += 1
        f -= 1
    return True
```

[3]:
```python
# Version récursive : un palindrome est soit une chaîne de longueur 0 ou 1,
#       soit une chaine dont le 1er et le dernier caractère sont identiques
#       ET la sous-chaine de 1 à N-1  est un palindrome
# (on peut dire aussi : un palindrome est soit une chaîne de longueur 0 ou 1,
#  soit un palindrome auquel est concaténée la même lettre à chaque bout)
def palindromeB(c):
    if len(c) <= 1:
        return True
    else:
        return c[0] == c[-1] and palindromeB(c[1:-1])
```

[4]:
```python
# Version "programmation structurée": pas de return ni de break dans la boucle
# on utilise un booléen pour garder la réponse en mémoire
def palindromeC(c):
    reponse = True
    for i in range(len(c)//2):
        if c[i] != c[-i-1]:
            reponse = False
    return reponse
```

[5]:
```python
# Version inspirée des vos copies: on commence par créer une copie inversée de␣
 ↪la chaîne initiale,
# puis on compare les deux copies.
# Le code est relativement simple mais l'algorithme est bien plus coûteux que␣
 ↪les versions précédentes.
def palindromeD(c):
    c_inv = ""
    for i in range(len(c)):
        c_inv = c[i] + c_inv
    return c == c_inv

# alternativement la boucle peut s'écrire
#   for i in range(-1, -longueur(c)-1, -1):
#       c_inv += c[i]
```

Pour vérifier que ces fonctions marchent toutes bien, je définis un jeu d'essai et une fonction qui peut tester en une seule fois toutes les fonctions que j'ai définies.

[8]:
```
# on définit une fonction de test pour les différentes implémentations
liste_mots = ["laval", "lavaliere", "mon ami", "navireerivan", "a", "", "aba",␣
 ↪"abb","abtotoba"]
def test_f_palindrome(f):
    for mot in liste_mots:
        print("%12s : %s" % (mot, f(mot)))
liste_fonctions = [palindrome, palindromeA, palindromeB, palindromeC,␣
 ↪palindromeD]
for f in liste_fonctions:
    test_f_palindrome(f)
    print("----")
```

```
       laval : True
   lavaliere : False
     mon ami : False
navireerivan : True
           a : True
             : True
         aba : True
         abb : False
    abtotoba : False
----
       laval : True
   lavaliere : False
     mon ami : False
navireerivan : True
           a : True
             : True
         aba : True
         abb : False
    abtotoba : False
----
       laval : True
   lavaliere : False
     mon ami : False
navireerivan : True
           a : True
             : True
         aba : True
         abb : False
    abtotoba : False
----
       laval : True
   lavaliere : False
     mon ami : False
```

```
navireerivan : True
           a : True
             : True
         aba : True
         abb : False
    abtotoba : False
----
       laval : True
   lavaliere : False
     mon ami : False
navireerivan : True
           a : True
             : True
         aba : True
         abb : False
    abtotoba : False
----
```

[ ]: