



```
print("Hello, world!")
```

FORMATION PYTHON : Les algorithmes de recherche

Ce chapitre a pour but d'initier aux algorithmes de recherche en Python.

Introduction

- ▶ Les algorithmes de recherche sont très utiles en informatique. Leur rôle est de :
 - ▶ Vérifier l'existence d'un ou de plusieurs éléments, satisfaisant une condition donnée, dans un tableau ;
 - ▶ Chercher le nombre d'occurrences d'un élément ;
 - ▶ Chercher la ou les positions d'un élément donné dans un tableau; etc.
- ▶ Il existe essentiellement deux stratégies de recherche :
 - ▶ Recherche *séquentielle* ou encore linéaire
 - ▶ Recherche *dichotomique*.

Plan

1. RECHERCHE SÉQUENTIELLE
2. RECHERCHE PAR DICHOTOMIQUE
3. EXERCICES (TD)

1. RECHERCHE SEQUENTIELLE

1. Recherche séquentielle

- ▶ La recherche séquentielle d'un élément dans un tableau consiste à parcourir le tableau séquentiellement jusqu'à trouver l'élément recherché ou atteindre la fin du tableau.
- ▶ La recherche peut se faire sur un tableau trié ou non, la différence réside dans le parcours du tableau et dans la condition d'arrêt de la recherche.

1. Recherche séquentielle

- ▶ On considère une liste L ou un tableau T et un objet e .

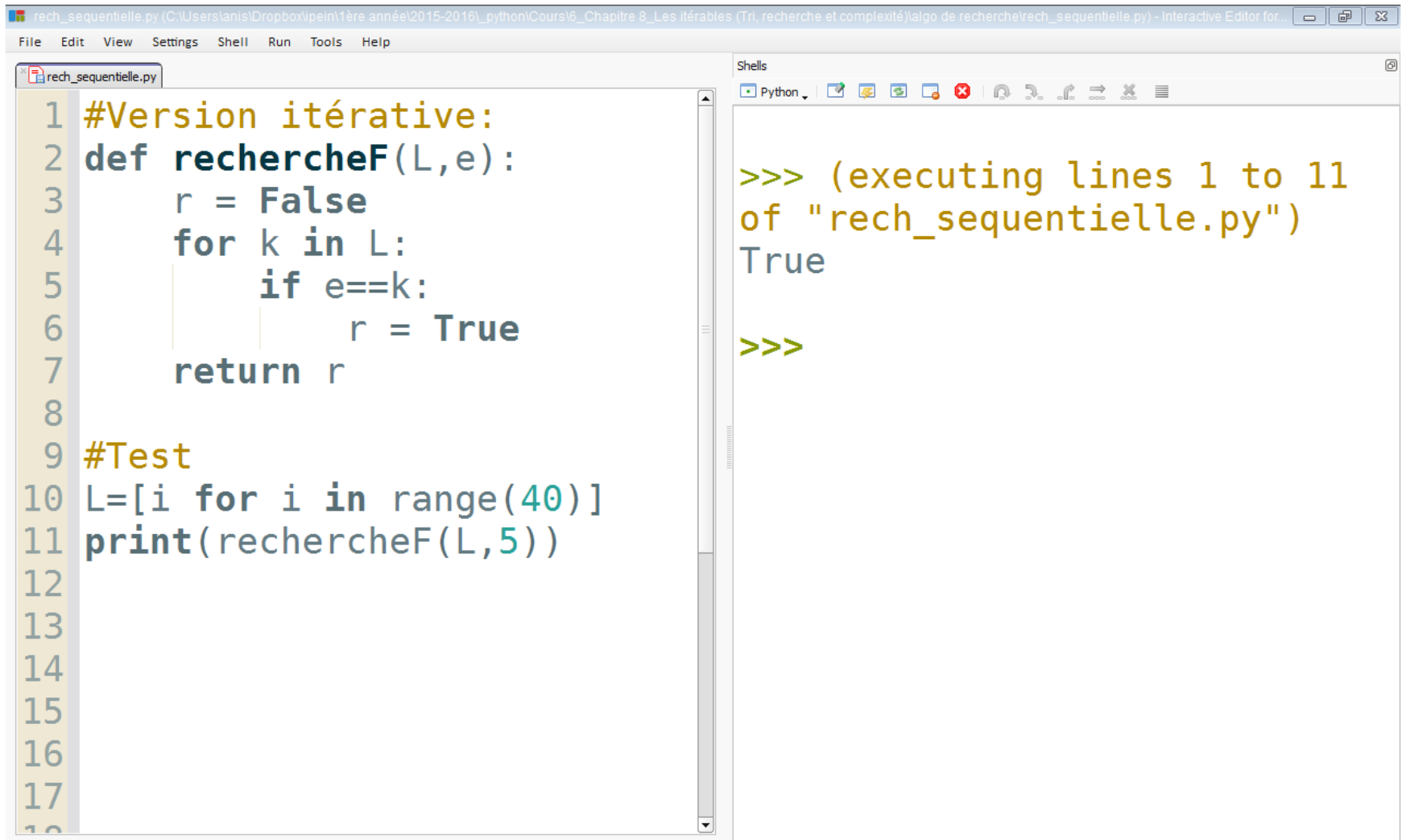
Problématique:

On souhaite tout d'abord de tester l'appartenance de e à L .

- ▶ Des primitives de Python réalisent ce travail ($e \text{ in } L$), mais notre objectif est de comprendre comment on les réalise.

1. Recherche séquentielle : V.I.

7



The image shows a screenshot of a Python IDE. The left pane displays the source code for a file named 'rech_sequentielle.py'. The code defines a function 'rechercheF(L, e)' that iterates through a list 'L' to find an element 'e'. It also includes a test section that creates a list of 40 integers and prints the result of the search function. The right pane shows the interactive shell output, which confirms that the function returns 'True' for the given input.

```
rech_sequentielle.py (C:\Users\anis\Dropbox\pein\1ère année\2015-2016_\python\Cours\6_Chapitre 8_Les itérables (Tri, recherche et complexité)\algo de recherche\rech_sequentielle.py) - Interactive Editor for...
File Edit View Settings Shell Run Tools Help

rech_sequentielle.py
1 #Version itérative:
2 def rechercheF(L,e):
3     r = False
4     for k in L:
5         if e==k:
6             r = True
7     return r
8
9 #Test
10 L=[i for i in range(40)]
11 print(rechercheF(L,5))
12
13
14
15
16
17
18
```

```
Shells
Python
>>> (executing lines 1 to 11
of "rech_sequentielle.py")
True
>>>
```

1. Recherche séquentielle

Complexité :

- ▶ Un appel à la procédure rechercheF(L,e) provoque n itérations, avec $n = \text{len}(L)$, dans tous les cas ce qui fait n comparaisons

if (e==L[k])
- ▶ $O(n)$: Complexité linéaire

1. Recherche séquentielle : EX1

Exercice :

- ▶ Écrire une fonction qui renvoie les deux plus grands éléments d'un tableau d'entiers. On supposera que le tableau est de longueur au moins 2 ; en revanche, on veillera à ne le parcourir qu'une seule fois.

1. Recherche séquentielle : EX1 (S)

10

```
ex_2max.py (C:\Users\anis\Dropbox\pein\1ère année\2015-2016_\python\Cours\Chapitre 8_Les itérables (Tri, recherche et complexité)\algo de recherche\ex_2max.py) - Interactive Editor for Python
File Edit View Settings Shell Run Tools Help

ex_2max.py
1 from numpy import *
2
3 def rech_2max(T):
4     """ renvoie les deux plus grands éléments
5         d'un tableau d'entiers """
6     #initialisations
7     n=len(T)
8     if(T[0]>T[1]):
9         max1,max2=T[0],T[1]
10    else:
11        max1,max2=T[1],T[0]
12
13    #parcours du tableau à partir de 3ème élément
14    for i in range(2,n):
15        if (T[i]>max1):
16            max2=max1
17            max1=T[i]
18        elif (T[i]<max1) and (T[i]>max2):
19            max2=T[i]
20
21    return max1,max2
22
23 #Appel
24 T=array([12,6,0,-4,13,8])
25 max1, max2 = rech_2max(T)
26 print("max1 = ",max1, "\nmax2 = ", max2)
27
28

Shells
Python
>>> (executing lines 1
to 27 of "ex_2max.py")
max1 = 13
max2 = 12
>>>
```

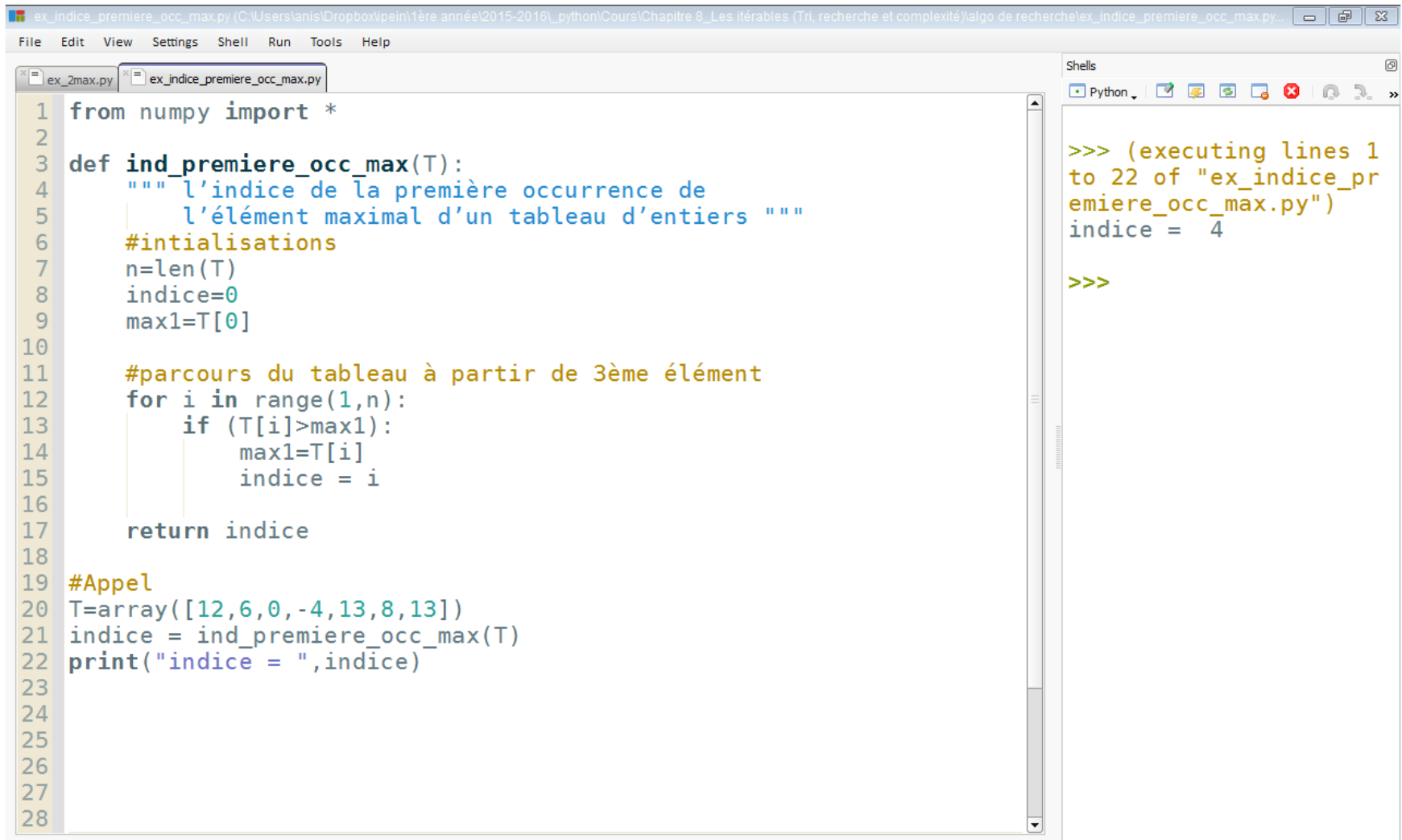
1. Recherche séquentielle : EX 2

Exercice :

1. Écrire une fonction qui renvoie l'indice de la première occurrence de l'élément maximal d'un tableau d'entiers.
2. Évaluer la complexité de cette fonction.

1. Recherche séquentielle :EX 2 (S)

12



The image shows a Python IDE window with two tabs: 'ex_2max.py' and 'ex_indice_premiere_occ_max.py'. The active tab contains the following Python code:

```
1 from numpy import *
2
3 def ind_premiere_occ_max(T):
4     """ l'indice de la première occurrence de
5         l'élément maximal d'un tableau d'entiers """
6     #initialisations
7     n=len(T)
8     indice=0
9     max1=T[0]
10
11     #parcours du tableau à partir de 3ème élément
12     for i in range(1,n):
13         if (T[i]>max1):
14             max1=T[i]
15             indice = i
16
17     return indice
18
19 #Appel
20 T=array([12,6,0, -4,13,8,13])
21 indice = ind_premiere_occ_max(T)
22 print("indice = ",indice)
23
24
25
26
27
28
```

The right-hand side of the IDE shows a 'Shells' window with the following output:

```
>>> (executing lines 1
to 22 of "ex_indice_pr
emiere_occ_max.py")
indice = 4

>>>
```

1. RECHERCHE DICHOTOMIQUE

2. Recherche dichotomique

- ▶ L'algorithme que nous proposons ici suppose que la liste dans laquelle on recherche un élément est **triée**.
- ▶ Quand la liste (tableau) est triée, la recherche d'un élément dans une liste peut être réalisée de manière plus efficace en appliquant le principe "*diviser pour régner*".
- ▶ L'idée est que l'on compare l'élément cherché au terme du milieu de la liste, ce qui conduit, tant qu'on ne le trouve pas, à le rechercher soit dans la première partie soit dans la seconde partie de la liste.

2. Recherche dichotomique : principe

Exemple : Chercher la valeur 9 dans le tableau [1, 3, 5, 6, 9, 12, 14].

La recherche s'effectue ainsi :

On cherche dans $a[0:7]$: 1 3 5 6 9 12 14 avec : $\text{len}(a)=7$

On compare x à $a[3]$: 1 3 5 6 9 12 14 9 > 6 \Rightarrow continuer à droite

On cherche x dans $a[4:7]$: 1 3 5 6 9 12 14

On compare x à $a[5]$: 1 3 5 6 9 12 14 12 > 9 \Rightarrow continuer à gauche

On cherche dans $a[4:4]$: 1 3 5 6 9 12 14

On compare $x=9$ avec $a[4]=9$:

1 3 5 6 9 12 14 9=9

La valeur 9 a été trouvée en seulement trois.

2. Recherche dichotomique : V.I.

16

```
rech_dichotomique_ver_iterative.py (C:\Users\anis\Dropbox\ipein\1ère année\2015-2016_python\Cours\6_Chapitre 8_Les itérables (Tri, recherche et complexité)\algo de recherche\rech_dichotomique_ver_iter...
File Edit View Settings Shell Run Tools Help

rech_dichotomique_ver_iterative.py
1 #Recherche dichotomique
2 #L : une liste triée
3 #Version itérative
4 def rechDicho (L,e):
5     a,b =0,len(L)-1
6     while a<=b:
7         m=(a+b)//2
8         if L[m]==e:
9             return True
10        elif L[m]<e:
11            a=m+1
12        else:
13            b=m-1
14        return False
15
16 L=[2,4,5,7,1,0,52,7,8]
17 print(rechDicho(L,4))
18

Shells
Python
>>> (executing lines 1 to
17 of "rech_dichotomique_
ver_iterative.py")
False
>>>
```


2. Recherche dichotomique : V.R.

```
#Recherche dichotomique dans une liste triée
```

```
def rechDicho (L,e):  
    a,b=0,len(L)  
    m=(a+b)//2  
    if(a==b==0):  
        |    return False  
    elif L[m]==e:  
        |    return True  
    elif L[m]>e:  
        |    return rechDicho(L[:m],e)  
    else:  
        |    return rechDicho(L[m+1:],e)
```

```
#Test : La liste (tableau) doit être trié
```

```
L=[2,4,5,6,7,8,9]  
print(rechDicho(L,59))
```

2. Recherche dichotomique : Exercice

19

Recherche d'un mot (une séquence) dans un texte

- ▶ Un problème classique en informatique consiste à rechercher, non pas une seule valeur, mais une *séquence* de valeurs dans un tableau. Cela revient à chercher une occurrence d'un tableau dans un autre ou, pour les chaînes de caractères, une occurrence d'un mot dans un texte.
- ▶ On souhaite donc écrire une fonction **recherche_mot** qui, étant donnés deux tableaux m et t , détermine la position de la première occurrence de m dans t , si elle existe, et qui renvoie **None** sinon.

Exemple : pour les tableaux $m=[1,2,3]$ et $t=[2,1,4,1,2,6,1,2,3,7]$,

La fonction **recherche_mot** renvoie 6 : [2, 1, 4, 1, 2, 6, **1**, 2, 3, 7]

2. Recherche dichotomique : Ex (S)

20

```
ex_recherche_mot_dichotomique.py (C:\Users\stanis\Dropbox\ipein\1ère année\2015-2016_\python\Cours\Chapitre 8_ Les itérables (Tri, recherche et complexité)\algo de recherche\ex_recherche_mot_dichotomi...
File Edit View Settings Shell Run Tools Help

ex_recherche_mot_dichotomique.py
1 from numpy import *
2
3 def recherche_mot(t,m):
4     """ t,m : deux tableaux
5         La fonction détermine la position de la première occurrence
6         de m dans t, si elle existe, sinon elle renvoie None """
7     #initialisations
8     n1 = len(t)
9     n2 = len(m)
10
11     for i in range(n1):
12         if t[i] == m[0] and i+n2<=n1:
13             for j in range(n2):
14                 if t[i+j]!=m[j]:
15                     break #le deuxième tableau de correspond pas
16             else:
17                 #tous les éléments de m existent dans t
18                 return i #retourne la première occurrence
19     return None
20
21 #Appel
22 t=[2,1,4,1,2,6,1,2,3,1,2,3,7,9]
23 m=[1,2,3]
24 resultat = recherche_mot(t,m)
25 print(resultat)
26
27
28
```

```
Shells
Python
>>> (executing lines 1 to 23 of "ex_recherche_mot_dichotomique.py")
6
>>>
```

2. Recherche dichotomique : Exercice

22

1. Modifier la fonction **recherche_mot** pour qu'elle retourne *toutes* les occurrences de *m* dans *t*.

2. Recherche dichotomique : Ex (S1)

23

```
ex_recherche_mot_dichotomique_all_occ.py (C:\Users\anis\Dropbox\1ère année\2015-2016_\python\Cours\Chapitre 8_Les itérables (Tri, recherche et complexité)\algo de recherche\ex_recherche_mot_d...
File Edit View Settings Shell Run Tools Help

ex_recherche_mot_dichotomique_all_occ.py
1 from numpy import *
2
3 def recherche_mot(t,m):
4     """ t,m : deux tableaux
5         La fonction détermine les positions de toutes les occurrences
6         de m dans t, si elle existe, sinon elle renvoie None """
7     #initialisations
8     n1 = len(t)
9     n2 = len(m)
10    occ=[]
11    for i in range(n1):
12        if t[i] == m[0] and i+n2<=n1:
13            for j in range(n2):
14                if t[i+j]!=m[j]:
15                    break #le deuxième tableau ne correspond pas
16            else:
17                #tous les éléments de m existent dans t
18                occ.append(i)
19    if(len(occ)>0):
20        return occ
21    return None
22
23 #Appel
24 t=[2,1,4,1,2,6,1,2,3,1,2,3,7,9]
25 m=[1,2,3]
26 resultat = recherche_mot(t,m)
27 print(resultat)
28
```

```
Shells
Python
>>> (executing lines 1 to 27 of "ex_recherche_mot_dichotomique_all_occ.py")
[6, 9]
>>>
```

2. Recherche dichotomique : Exercice

24

1. Ecrire une fonction qui vérifie qu'une chaîne de caractères est un palindrome, c'est-à-dire qu'elle est identique qu'on la lise de gauche à droite ou de droite à gauche.
2. Adapter cette fonction pour qu'elle ne tienne pas compte des espaces ni des signes de ponctuation.

2. Recherche dichotomique : Ex (S)

25

```
ex_palindrome1.py (C:\Users\anis\Dropbox\ipe\1ère année\2015-2016\python\Cours\Chapitre 8_Les itérables (Tri, recherche et complexité)\algo de recherche\ex_palindrome1.py) - Interactive Editor for Python
File Edit View Settings Shell Run Tools Help

ex_palindrome1.py
1 def palindrome1(m):
2     """ Vérifie qu'une chaîne de caractères est un palindrome,
3         c'est-à-dire qu'elle est identique qu'on la lise
4         de gauche à droite ou de droite à gauche """
5     l=list(m)
6     l1=list(m)
7     l.reverse()
8     return l==l1
9
10 #test
11 m="radar"
12 print(palindrome1(m))
13
14 def palindrome2(m):
15     """ ne tienne pas compte des espaces ni des
16         signes de ponctuation """
17     l,l1=[],[]
18     for i in m:
19         if (i.isalpha()):
20             l.append(i)
21             l1.append(i)
22     l.reverse()
23     return l==l1
24 #test
25 m="radar"
26 print(palindrome2(m))
27
28

Shells
Python
>>> (executing lines 1
to 26 of "ex_palindrom
e1.py")
True
True
>>>
```