Les listes

Lists

Syntax

To create a list: $[value_1, value_2, \ldots]$

To access an element: *listReference*[*index*]

Creates an empty list

Creates a list with initial values

Name of list variable

```
moreValues = []
values = [32, 54, 67, 29, 35, 80, 115]
```

Initial values

Use brackets to access an element.

```
values[i] = 0
element = values[i]
```

Les listes

1. Les listes - première approche

Nous avons vu que les listes étaient un cas particulier d'un type de données plus général que l'on appelle des **données composites.**
Une liste (un type d'objet) est une **collection ordonnée et modifiable** d'éléments éventuellement hétérogènes. Les éléments sont séparés par des **vigules et entourés de crochets**.

Exemple : la variable jour est une liste (ojet de type liste) ici :

```
jour = ['lundi','mardi','mercredi',1800,20.357,'jeudi','vendredi']
print(jour)


>>>
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
>>>
```

1.1. Accès aux éléments par l'index

Comme on peut le constater, les éléments individuels qui constituent la liste peuvent être de **types variés.** Comme pour les chaînes de caractères, les éléments d'une liste sont ordonnés toujours dans le même ordre et l'on peut donc accéder à chacun d'entre eux par son **index.**

```
jour = ['lundi','mardi','mercredi',1800,20.357,'jeudi','vendredi']
print(jour)
print(jour[2])
print(jour[4])

>>>
['lundi', 'mardi', 'mercredi', 1800, 20.357, 'jeudi', 'vendredi']
mercredi
20.357
>>>
```

Mais à la différence de ce qui se passe pour les chaînes, il est possible de **changer les éléments individuels** d'une liste:

```
jour = ['lundi','mardi','mercredi',1800,20.357,'jeudi','vendredi']
jour[3]=jour[3]+47
print(jour)

jour[3]="juillet"
print(jour)

>>>
['lundi', 'mardi', 'mercredi', 1847, 20.357, 'jeudi', 'vendredi']
['lundi', 'mardi', 'mercredi', 'juillet', 20.357, 'jeudi', 'vendredi']
>>>
```

1.2. Nombre d'éléments dans une liste

La fonction intégrée len() s'applique aussi aux listes. Elle renvoie le nombre d'éléments présents dans la liste:

```
>>> print(len(jour))
7
```

Les listes

1.3. Suppression et/ou ajout d'un élément d'une liste

Une autre fonction intégrée permet de supprimer d'une liste un élément quelconque. Il s'agit de la fonction del().

```
jour = ['lundi','mardi','mercredi',1800,20.357,'jeudi','vendredi']
del(jour[4])
print(jour)

>>>
['lundi', 'mardi', 'mercredi', 1800, 'jeudi', 'vendredi']
>>>
```

Pour ajouter un élément à une liste, il faut considérer que la liste est un objet. On va utiliser l'instruction append() (qui fait uniquement un ajout en fin de liste).

```
jour.append("samedi")
print(jour)
['lundi', 'mardi', 'mercredi', 1800, 'jeudi', 'vendredi', 'samedi']
>>>
```

Exercice:

☐ Veuillez analyser le petit script ci dessous et commenter son fonctionnement.

```
jour = ['dimanche','lundi','mardi','mercredi','jeudi','vendredi','samedi']
a,b = 0,0
while a<8:
    a=a+1
    b=a%7
    print(a,jour[b])
```

```
1 lundi
2 mardi
3 mercredi
4 jeudi
5 vendredi
6 samedi
7 dimanche
8 lundi
>>>
```

1.4. Découpage en tranches nom_liste[*deb* : *fin*]

Une tranche découpée dans une liste est toujours elle même une liste. On utilise un **début : une fin** pour isoler la tranche voulue.

Exemple :

```
nombres=[5,38,10,25]
mots=["jambon","fromage","confiture","chocolat"]
stuff=[5000,"Brigitte",3.1416,["Albert","René",1947]]

print(nombres[2])
print(nombres[1:3])
print()
print(mots[2:3])
print()
print(stuff[2:])
print(stuff[4])
print(stuff[:2])
print(stuff[-1])
print(stuff[-2])
```

```
>>>
10
[38, 10]

['confiture']

[3.1416, ['Albert', 'René', 1947]]
Traceback (most recent call last):
  File "F:/2013 2014 PTSI/INFORMATIQUE/INFO PTSI 2014-2015/S15-16. Listes/Cours/
5. découpage tranches.py", line 11, in <module>
    print(stuff[4])
IndexError: list index out of range
>>>
```

ATTENTION: l'élément de rang deb est inclus, pas celui de rang fin (idem range).

Les listes

1.5. Appel d'une liste à l'utilisateur

On utilise l'instruction déjà vue input() mais attention avec les listes !!!

```
>>> liste=input("Entrez votre liste:")
Entrez votre liste:[1,2,3]
>>> print(liste)
[1,2,3]
>>> type(liste)
<class 'str'>
>>> liste[1]
'1'
>>> liste[0]
'['
>>>
```

1.6. Evaluation des éléments d'une liste

On préfèrera donc cette méthode par rapport à l'appel input ( ) : vous pouvez déterminer la nature d'un élément d'une liste par l'instruction: eval(input()) ou list().

```
liste=eval(input("entrez des nombres: "))
print(liste)
n=liste[1]*liste[2]
print(n)


liste2=input("entrez des nombres: ")
print(liste2)
n=liste2[1]*liste2[2]
print(n)
```

```
>>>
entrez des nombres: [12,3,7,8]
[12, 3, 7, 8]
21
entrez des nombres: [12,3,7,8]
[12,3,7,8]
Traceback (most recent call last):
  File "F:\2013 2014 PTSI\INFORMATIQUE\INFO PTSI 2014-2015\S15-16. Listes\Cours
9. eval input.py", line 9, in <module>
    n=liste2[1]*liste2[2]
TypeError: can't multiply sequence by non-int of type 'str'
>>>
```

1.7. Concaténation et multiplication et autres opérations sur les listes

On peut appliquer aux listes les opérateurs + et *.

```
fruits=["orange","citron"]
légumes=["poireau","oignon","tomate"]

print(fruits+légumes)
print(fruits*3)

print()
#rappel chaines caractères

fruit="orange"
légume="poireau"
print(fruit*3)
print(fruit+légume)
```

```
>>>
['orange', 'citron', 'poireau', 'oignon', 'tomate']
['orange', 'citron', 'orange', 'citron', 'orange', 'citron']

orangeorangeorange
orangepoireau
>>>
```

```
liste1=[2,7,15]
liste2=[3,5,8]

liste3=liste1+liste2

print(liste1)
print(liste2)
print(liste3)
```

```
>>>
[2, 7, 15]
[3, 5, 8]
[2, 7, 15, 3, 5, 8]
>>>
```

Ici nous avons une somme de 2 listes avec des entiers (en considérant que la 1ère est la plus grande)

```
def somme_liste(list1, list2):

    list3 = list1[:]
    for i,val in enumerate(list2):
        list3[i] = list3[i] + val

    return list3


liste1 = eval(input("entrer liste 1: "))
liste2 = eval(input("entrer liste 2: "))

liste3 = somme_liste(liste1, liste2)
print(liste3)
```

Mais il y a plus simple avec une **liste en compréhension** avec l'instruction **zip ( )** (très pratique pour concours !!) :

```
# liste en comprehension avec zip()

def somme_liste(list1, list2):

    liste3 = [x+y for x,y in zip(liste1, liste2)]
    return liste3


liste1 = eval(input("entrer liste 1: "))
liste2 = eval(input("entrer liste 2: "))

liste3 = somme_liste(liste1, liste2)
print(liste3)
```

Enfin, pour la multiplication, l'opérateur * est particulièrement utile pour créer une liste de n éléments identiques :

```
m=[0]*7
print(m)
```

```
[0, 0, 0, 0, 0, 0, 0]
```

1.8. Test d'appartenance

Vous pouvez déterminer si un élément fait partie d'une liste ou non avec l'instruction in.

```
fruits=["orange","citron"]
légumes=["poireau","oignon","tomate"]

v="tomate"
if v in légumes:
    print("ok")
```

Les listes

1.9. Listes en compréhension

Pour avoir des codes plus compacts et réduire le temps d'exécution, on peut créer des listes en compréhension comme dans les exemples ci-dessous.

Exemple :

Exemple 1.

```
# liste en compréhension, exemple 1
L = [2 * k + 1 for k in range(10,17)] # k prend les valeurs 10, 11 ,... , 15 ,16 ,␣
 ↪soit 17-10 = 7 valeurs distinctes
print(L)
```

```
[21, 23, 25, 27, 29, 31, 33]
```

Exemple 2.

Entiers impairs divisibles par 7.

```
# liste en compréhension, exemple 2
L = [2*k+1 for k in range(50) if (2*k+1) % 7 == 0] # attention
                        # à la parenthèse avant le modulo "%"!
print(L) # entiers impairs multiples de 7
```

```
[7, 21, 35, 49, 63, 77, 91]
```

Rappel : `a % b` désigne le reste de la division entière de `a` par `b` (a modulo b).

2. Les listes sont des objets

Sous Python, les listes sont des objets à part entière, et vous pouvez donc leur appliquer un certain nombre de METHODES particulièrement efficaces. En voici quelques unes:

```
nombres=[17,38,10,25,72]

print(nombres.sort())       # trier la liste

nombres.append(12)          # ajouter élément à la fin liste
print(nombres)

nombres.reverse()           # inverser l'ordre des éléments
print(nombres)

print(nombres.index(17))    # retrouver index d'un élément (pas find)

nombres.remove(38)          # enlever (effacer) un élément
print(nombres)
```

Attention, ne marche pas direct avec print !!!

```
None
[10, 17, 25, 38, 72, 12]
[12, 72, 38, 25, 17, 10]
4
[12, 72, 25, 17, 10]
>>>
```

Les listes

On dispose aussi de l'instruction del, qui vous permet d'effacer un ou plusieurs éléments à partir de leurs index:

```
nombres=[17,38,10,25,72]

del nombres[2]
print(nombres)

del nombres[1:2]
print(nombres)
```

```
>>>
[17, 38, 25, 72]
[17, 25, 72]
>>>
```

Notez bien la différence entre la méthode remove ( ) et l'instruction del. del travaille avec un index ou une tranche d'index, tandis que remove ( ) recherche une valeur.

En voici quelques autres ...

- nom_liste.**extend**(liste2)
  → ajoute tous les éléments de la liste2 à la fin de liste
- nom_liste.**insert**(pos, val) → insère *val* en position *pos*
- nom_liste.**count**(valeur) → renvoie le nombre de fois que valeur apparaît dans la liste
- nom_liste.**pop**() ou nom_liste.**pop**(pos)
  → enlève l'élément présent à la position *pos* donnée dans la liste et le renvoie.
  → Si aucun indice n'est spécifié, renvoie et supprime le dernier élément de la liste

```
nombres=[17,38,10,25,72]
chiffres=[21,13]

nombres.extend(chiffres)
print(nombres)

nombres.insert(2,17)
print(nombres)

print(nombres.count(17))

nombres.pop()
print(nombres)
nombres.pop(3)
print(nombres)
```

```
>>>
[17, 38, 10, 25, 72, 21, 13]
[17, 38, 17, 10, 25, 72, 21, 13]
2
[17, 38, 17, 10, 25, 72, 21]
[17, 38, 17, 25, 72, 21]
>>>
```

Construction	Meaning
`a = []`	initialize an empty list
`a = [1, 4.4, 'run.py']`	initialize a list
`a.append(elem)`	add `elem` object to the end
`a + [1,3]`	add two lists
`a.insert(i, e)`	insert element `e` before index `i`
`a[3]`	index a list element
`a[-1]`	get last list element
`a[1:3]`	slice: copy data to sublist (here: index 1, 2)
`del a[3]`	delete an element (index 3)

Construction	Meaning
`a.remove(e)`	remove an element with value `e`
`a.index('run.py')`	find index corresponding to an element's value
`'run.py' in a`	test if a value is contained in the list
`a.count(v)`	count how many elements that have the value `v`
`len(a)`	number of elements in list `a`
`min(a)`	the smallest element in `a`
`max(a)`	the largest element in `a`
`sum(a)`	add all elements in `a`
`sorted(a)`	return sorted version of list `a`
`reversed(a)`	return reversed sorted version of list `a`
`b[3][0][2]`	nested list indexing
`isinstance(a, list)`	is `True` if `a` is a list
`type(a) is list`	is `True` if `a` is a list

Les listes

3. Technique de slicing avancé pour modifier une liste

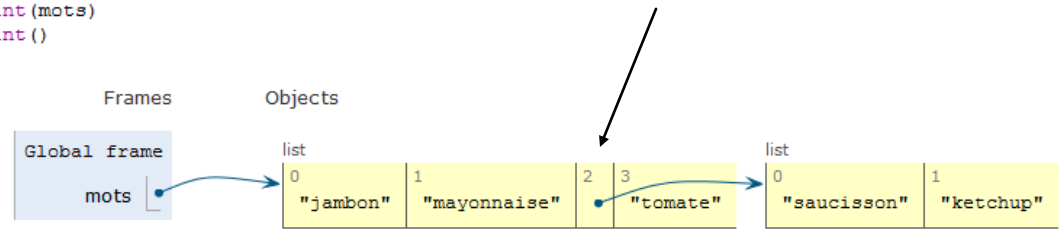
Comme nous venons de le voir, on peut utiliser l'instruction del ou append pour ajouter ou supprimer des éléments d'une liste. Mais vous pouvez également obtenir les mêmes résultats à l'aide d'un seul opérateur [ ]. L'utilisation de cet opérateur est un peu plus délicate mais elle permet davantage de souplesse:

Insertion d'un ou plusieurs éléments n'importe où dans une liste

```
mots=["jambon","14","confiture","18"]
print(mots)
print()

mots[2]="miel"              # on a déjà vu ça pour remplacer un élément dans une liste
print(mots)

mots[2:2]=["miel"]            # autre façon de faire (ajoute à l'emplacement 2 l'élément "miel")
print(mots)


mots[5:5]=["saucisson","22"]  # ajoute en le créant à l'emplacement 5, 2 éléments "saucisson","ketchup"
print(mots)
```

Suppression / remplacement d'éléments

```
mots[5:5]=["saucisson","22"]  # ajoute en le créant à l'emplacement 5, 2 éléments "saucisson","ketchup"
print(mots)

mots[2:5]=[]                  # supprime les éléments des emplacements 2,3,4
print(mots)

mots[1:3]=["salade"]          # remplace les éléments aux emplacements 1 et 2 par l'élément "salade"
print(mots)

mots[1:]=["mayonnaise","poulet","tomate"] # idem
print(mots)


mots[2]=["saucisson","ketchup"]    # attention, ici on rajoute une liste à l'emplacement 2
print(mots)
print()
```



Autres exemples de slicing :

- `L[a:b]` désigne la sous-liste formée des éléments de `L` dont l'indice est compris entre a et b-1.
- `L[-1]` désigne le dernier élément, `L[-2]`, l'avant dernier, etc…
- `L[a:b:step]` désigne la liste des éléments dont l'indice est a pour valeurs :

  a, a+step, a+2step, a+ 3step, …, a+k.step. Jusqu' à $a+kstep < b$

  Attention : **b n'est jamais atteint!**

Exemples de sciling sur des listes.

```
L=['A','B','C','D','E','F','G','H']
print('1: ', L[6])   # affiche le 7ème élément de la liste
print('2: ', L[-3])  # avant-avant dernier élément
print('3: ', L[2:])  # on retire les 2 premiers éléments
print('4: ', L[0:3]) # liste formée des 3 premiers éléments
                     # (k = 0, 1 et 2)
```

```
print('5: ', L[0:6:2]) # liste obtenue en parcourant les éléments
                       # de 2 en 2 (L[6] étant exclu)
print('6: ', L[-1::-1]) # liste parcourue en sens inverse
print('7: ', L[-1::-2]) # liste parcourue en sens inverse de 2 en 2
print('8: ', L[:])      # liste de tous les éléments, peut servir à
                        # faire une copie L2 = L1[:]
                        # mais il vaut mieux écrire L2 = L1.copy()
```

```
1:  G
2:  F
3:  ['C', 'D', 'E', 'F', 'G', 'H']
4:  ['A', 'B', 'C']
5:  ['A', 'C', 'E']
6:  ['H', 'G', 'F', 'E', 'D', 'C', 'B', 'A']
7:  ['H', 'F', 'D', 'B']
8:  ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H']
```

Remarque : `L2 = L1[ : ]` est équivalent à `L2 = L1.copy()`

4. Création d'une liste de nombres à l'aide de la fonction range()

Si vous devez manipuler des séquences de nombres, vous pouvez les créer très aisément à l'aide de la fonction range(). Elle renvoie une séquence d'entiers que vous pouvez utiliser directement, ou convertir en une liste avec la fonction list().

```
print(list(range(10)))

print(list(range(5,13)))

print(list(range(3,16,3)))

print(list(range(10,-10,-3)))
```

5. Parcours d'une liste à l'aide de for, range() et len()

L'instruction for est idéale pour parcourir une liste de caractères:

```
prov=["la","raison","du","plus","fort","est","toujours","la","meilleure"]
for mot in prov:
    print(mot,end=" ")

>>>
la raison du plus fort est toujours la meilleure
```

Les listes

Si vous voulez parcourir une gamme d'entiers, la fonction range() s'impose:

```
for n in range(10,18,3):
    print(n,n**2,n**3)
```

```
10 100 1000
13 169 2197
16 256 4096
>>>
```

Il est très pratique de combiner range() et len() pour obtenir automatiquement tous les indices d'une séquence:

```
fable=["maitre","corbeau","sur","son","arbre","perché"]
for index in range(len(fable)):
    print(index,fable[index])
```

```
0 maitre
1 corbeau
2 sur
3 son
4 arbre
5 perché
```

Autre méthode de parcours pour afficher le rang et l'élément d'une liste :

```
for indice, valeur in enumerate(ma_liste):
    instructions
```

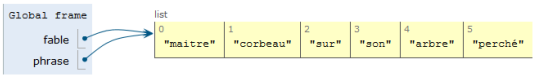
```
top3_petrole=["Arabie Saoudite","Russie","Etas-unis"]
for rg,pays in enumerate(top3_petrole):
    print("pays producteur de petrole de rang",rg,":",pays)
```

```
>>>
pays producteur de petrole de rang 0 : Arabie Saoudite
pays producteur de petrole de rang 1 : Russie
pays producteur de petrole de rang 2 : Etas-unis
>>>
```

6. Copie de listes

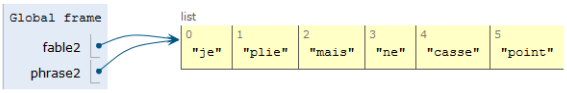
Considérons que vous disposez d'une liste fable que vous souhaitez copie dans une nouvelle variable phrase. Vous tapez ceci:

```
fable=["maitre","corbeau","sur","son","arbre","perché"]

phrase=fable
```



En procédant ainsi, vous ne créez pas une véritable copie mais un alias. A la suite de cette instruction, il n'existe toujours qu'une seule liste dans la mémoire du PC. On créé juste une nouvelle référence vers cette liste:

```
fable2=["je","plie","mais","ne","romps","point"]
phrase2=fable2
fable2[4]="casse"
print(phrase2)
```



Remarque sur les performances: temps de calcul

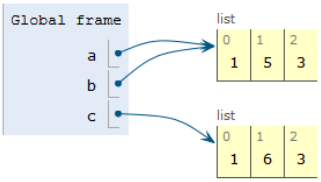
Il convient de mener une réflexion sur le temps de calcul lié à l'utilisation des listes ou tableaux. Ce temps de calcul est directement lié à la façon dont ces structures sont implémentées dans la mémoire de la machine.

Pour les listes, **la machine stocke en mémoire les éléments de façon chaînée**. Partant d'un élément appelé tête de la liste, chaque élément a connaissance de l'adresse où se trouve l'élément suivant. Pour accèder au $i^{ème}$ élément, on parcourt la chaîne depuis la tête jusqu'à l'élément désiré.

Les listes

Différence importante entre copie d'une valeur et copie d'une adresse:

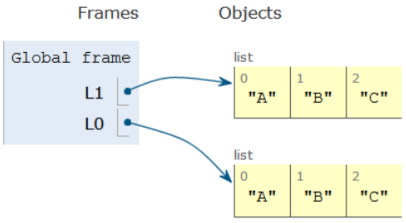
```
a=[1,2,3]
b=a          # b se voit affecté de la même adresse que a
             # a et b sont 2 noms du même objet

b[1]=5       # b vaut maintenant [1,5,3], mais a aussi !!

c=a[:]      # ici, on copie dans c les éléments de a
            # c est un clone de a mais physiquement distinct de celui-ci

c[1]=6      # c vaut maintenant [1,6,3] mais a et b n'ont pas changés
```



Il convient donc de bien comprendre si l'instruction ← copie une valeur ou copie une adresse. Le comportement des variables est totalement différent dans les deux cas.

Pour copier une liste, on utilise la méthode nom_liste.copy().

```
L1 = ['A', 'B', 'C']
L0 = L1.copy() # L1.copy() crée un nouvel objet.
               # L0 y fait ensuite référence
```



Avec la méthode `L1.copy()` un nouvel objet est créé.

7. Nombres aléatoires

La plupart des programmes d'ordinateur font exactement la même chose chaque fois qu'on les exécute. On les dits déterministes. Il existe cependant des techniques mathématiques permettant de simuler plus ou moins bien l'effet du hasard.

Dans le module random, python propose toute une série de fonctions permettant de générer des nombres aléatoires qui suivent différentes distributions mathématiques.

Vous pouvez importer toutes les fonctions par:

```
>>> from random import *
```

La fonction random du module random permet de créer une liste de nombres réels aléatoires, de valeur comprise entre 0 et 1. L'argument à fournir est la taille de la liste:

```
from random import random

def list_aleat(n):
    s=[0]*n
    for i in range(n):
        s[i]=random()
    return s
n=int(input("combien de chiffres voulez vous tirer au hasard ? "))
print(list_aleat(n))
print(list_aleat(n))
```

```
combien de chiffres voulez vous tirer au hasard ? 3
[0.3881008773284369, 0.21274313231929975, 0.3215857826427073]
[0.025765853444587483, 0.5238918568131622, 0.3548204367403971]
>>>
```

Attention: nous avons d'abord construit une liste de 0 de taille n, puis remplacé les 0 par des nombres aléatoires

Lorsque vous développerez des projets personnels, il vous arrivera fréquemment de souhaiter disposer d'une fonction qui permette de tirer au hasard un nombre entier entre certaines limites.

Les listes

Vous pouvez utiliser la fonction randrange() du module random. Elle peut être utilisée avec 1,2,3 arguments.

Exemple :

```
>>> from random import*
>>> randrange(6)                    renvoie 0<entier<5
5
>>> randrange(2,8)                  renvoie 2<entier<7
2
>>> randrange(3,13,3)               renvoie  entier dans série 3,6,9,12
3

from random import randrange

for i in range(15):
    print(randrange(100),end=" ")
```

```
13 34 37 84 14 99 58 18 25 52 11 19 25 69 4
>>>
```

Exercice :

☐ Ecrivez un script qui tire au hasard une carte à jouer (jeu de 52 cartes). Le nom de la carte tirée doit être affiché en clair. L'utilisateur devra taper <c> puis <enter> pour tirer une carte, et <enter> seulement s'il souhaite arrêter.
PS: on vous impose d'utiliser break pour sortir du programme et pop() pour extraire des listes la carte...

```
>>>
frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter: c
7 de trèfle
frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter: c
2 de trèfle
frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter: c
dame de trèfle
frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter: c
valet de trèfle
frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter:
>>>
```

Solution 1 :

```
from random import *

bois=["pique","trèfle","carreau","coeur"]
valeurs=["2","3","4","5","6","7","8","9","10","valet","dame","roi","as"]

b=[]
v=[]

# tirage au hasard:

while 1:
    k=input("frappez <c> puis <enter> pour tirer une carte et <enter> pour arrêter: ")
    if k=="":
        break
    v.append(valeurs[randrange(13)])
    b.append(bois[randrange(4)])
    print(v.pop(0),"de",b.pop(0))
```

Solution 2 :

```
from random import*

bois=["coeur","trèfle","pique","carreau"]
valeurs=["2","3","4","5","6","7","8","9","10","valet","dame","roi","as"]


#Traitement:
a=str(input("Frappez <c> puis <Entrer> pour tirer une carte, <enter> pour arrêter: "))

while a=="c":
    b=randrange(0,4)
    v=randrange(0,13)

#Sortie

    print(valeurs[v],end=" ")
    print("de",bois[b])
    a=str(input("Frappez <c> puis <Entrer> pour tirer une carte, <enter> pour arrêter: "))
```

Solution 3 :

```
from random import *

bois=["pique","trèfle","carreau","coeur"]
valeurs=["2","3","4","5","6","7","8","9","10","valet","dame","roi","as"]

# construction de la liste des 52 cartes

carte=[]
for b in bois:
    for v in valeurs:
        carte.append("{0} de {1}".format(v,b))

# tirage au hasard:

while 1:
    k=input("frappez <c> puis <enter> pour tirer une carte, <enter> pour terminer: ")
    if k=="":
        break
    r=randrange(52)    # tirage au hasard d'un entier entre 0 et 51
    print(carte[r])
```

montrer avec tutor qu'il créé une liste énorme !! pas optimisé