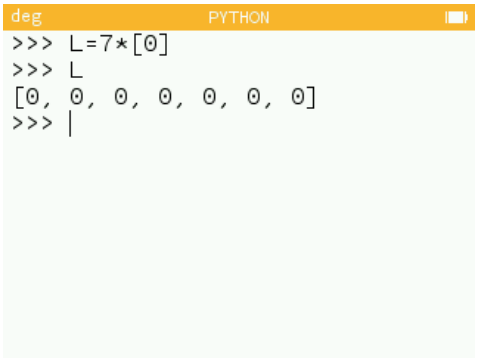
Introduction aux listes en Python

Une liste en Python est une structure de données qui peut contenir n'importe quel type de données (nombre entier, nombre flottant, chaîne de caractères, liste). Les éléments d'une liste sont rangés et séparés par des virgules. Le premier élément de la liste L a pour rang 0 ; il est noté `L[0]`.

Déclaration et initialisation

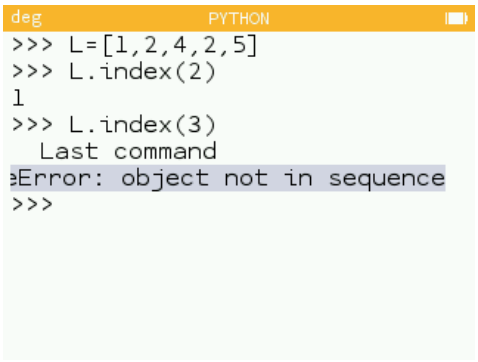
Une liste L peut être simplement déclarée par l'instruction `L=list()` ou `L=[]`. Elle existe alors mais ne contient aucun élément. On peut aussi la déclarer et l'initialiser en écrivant explicitement ses éléments : `L=[2,3,5,7,11,13,17,19,23,29]`.
On peut encore déclarer et initialiser une liste avec N éléments égaux à 0 en écrivant `L=N*[0]`.



Il y a bien d'autres façons de déclarer et d'initialiser une liste, par exemple avec la fonction `range()` : l'instruction `L=list(range(1,10))` conduit à la liste `L=[1,2,3,4,5,6,7,8,9]`.

Accès et modification d'un élément d'une liste

Lorsqu'une liste `L` a été créée, on peut accéder à l'élément de rang R en écrivant `L[R]` et le modifier ; en commençant par la fin : `L[-1]` est le dernier élément, `L[-2]` l'avant-dernier.

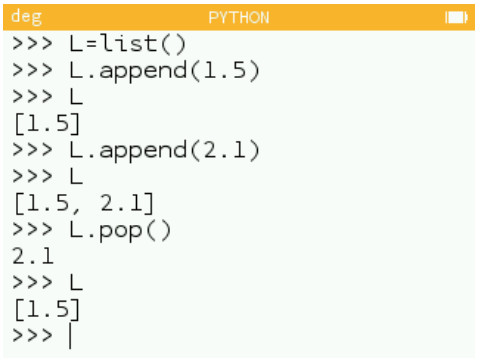


On peut aussi accéder au rang d'un élément : `L.index(E)` renvoie le premier rang de l'élément E.
Attention : s'il n'y a pas l'élément ou le rang demandé, ces instructions provoquent une erreur !

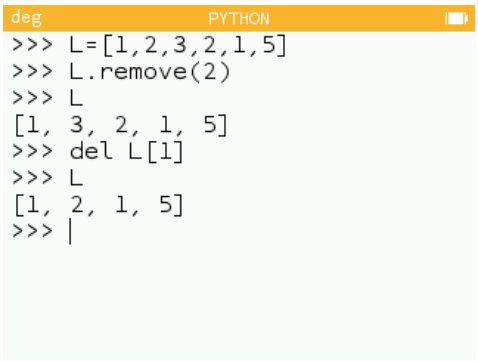
On peut extraire une liste E d'une liste L à partir d'un rang d jusqu'à un rang f en écrivant `E=L[d:f]`. Si un des rangs est omis, les valeurs par défaut sont d=0 (le début) et f=len(t) (la fin). Si par exemple, `L=[12,5,7]` alors `L[1:2]` renvoie `[5]`, `L[:2]` renvoie `[12,5]` et `L[1:]` renvoie `[5,7]`.

Ajout et suppression d'un élément dans une liste

On peut ajouter un élément E en dernière position dans une liste L en écrivant `L.append(E)`. Après `L=[]`, l'instruction `L.append(1.5)` conduit à avoir `L=[1.5]`. Si on écrit encore `L.append(2.1)`, on aura `L=[1.5,2.1]`. Pour supprimer le dernier élément d'une liste L, écrire `L.pop()`. Cette instruction renvoie l'élément supprimé qui peut alors être utilisé. `L.pop(R)` supprime et renvoie l'élément de rang R.



La méthode `insert(R,E)` insère l'élément E au rang R, en décalant tous les éléments suivants d'un rang.
Si `L = [1,2,3]`, l'instruction `L.insert(2,5)` insère l'élément 5 au rang 2, ce qui conduit à avoir `L=[1,2,5,3]`.



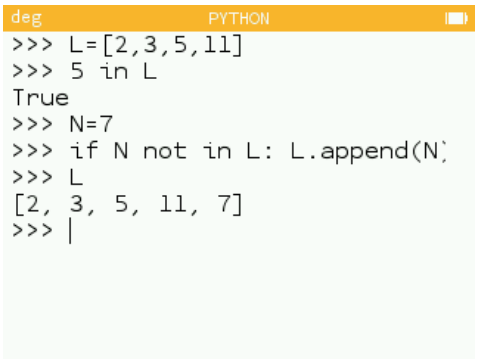
On peut aussi, avec `L.remove(E)`, enlever la première occurrence de l'élément E de la liste L. L'instruction `del L[R]` supprime l'élément de rang R, de la liste L.

Longueur d'une liste

Très utile pour éviter les erreurs de rang, l'instruction `len(L)` donne la longueur de la liste L (son nombre d'éléments).
Avec la liste `J=["Lu","Ma","Me","Je","Ve","Sa","Di"]`, l'instruction `len(J)` renvoie 7. Si j'ai une liste de listes, par exemple `M=[[0,0],[0,1],[1,0]]`, l'instruction `len(M)` renvoie 3 (c'est la longueur de la liste `M`), par contre `len(M[0])` renvoie 2 (c'est la longueur de la liste `M[0]`).

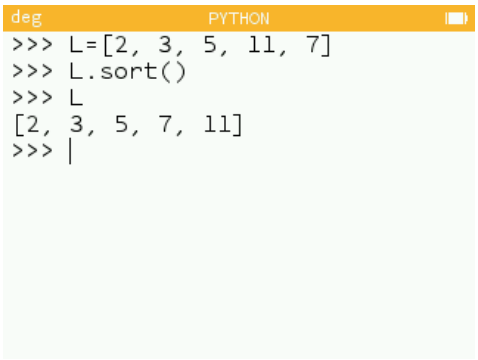
Test de la présence d'un élément dans une liste

Le mot-clé `in` permet de tester la présence d'un élément dans une liste. `N in L` renvoie `True` si N est un élément de la liste et `False` sinon. On peut tester si un élément N n'est pas dans la liste L en écrivant `N not in L`.



L'instruction `L.count(E)` donne le nombre d'occurrences d'un élément E dans une liste L.

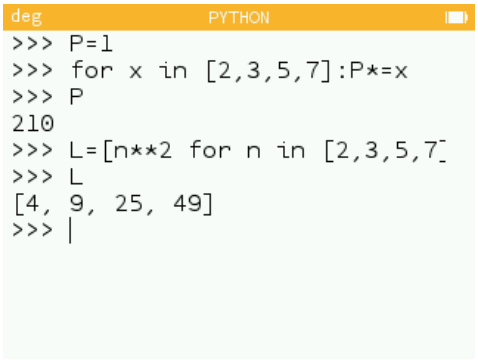
Tri d'une liste



Une liste L peut être triée dans l'ordre alphanumérique croissant avec l'instruction `L.sort()`. On obtiendra le tri dans l'ordre inverse en utilisant la méthode `reverse()`.

Parcourir une liste

L'instruction `for x in L` : est une déclaration de boucle bornée qui donne successivement à x la valeur des éléments de la liste L. Si P=1, l'instruction `for x in [2,3,5,7] : P*=x` conduit à P=210 (la syntaxe `P*=x` condense `P=P*x`). On peut parcourir une liste pour en créer une autre avec ce type de déclaration qui conduit à `L=[4,9,25,49]` : `L=[n*n for n in [2,3,5,7]]`.



On peut même effectuer un filtrage avec une condition `L=[n*n for n in [1,2,3,4,5] if n%2==1]` conduit à `L=[1,9,25]`.

Choix d'un élément aléatoire dans une liste

Cette possibilité est offerte par le module `random` qu'il faut donc importer au préalable par l'instruction `from random import choice` (ou `from random import *`).
Une liste L contenant au moins deux éléments, on en tire un N au hasard en écrivant `N=choice(L)`. Si `L=[1,10,100,1000]` alors `choice(L)` renvoie un élément au hasard de L : 1, 10, 100 ou bien 1000.

Exercice

Écrire une fonction `elimine(L)` qui supprime les doublons d'une liste : `elimine([1,2,2,1,2,1])` renvoie `[1,2]`.

On peut parcourir la liste `L` de départ et ajouter dans une liste `P` les éléments de L s'ils n'y sont pas déjà.

```
def elimine1(L):
  P=[]
  for e in L:
    if e not in P:
      P.append(e)
  return P
```

Une seconde option serait, pour chaque élément de L, de se demander s'il est présent à un rang supérieur et dans ce cas supprimer les occurrences redondantes (on n'a pas alors besoin d'une liste P).

```
def elimine2(L):
  for e in L:
    if L.count(e)>1:
      for i in range(L.count(e)-1):
        L.remove(e)
  return L
```

Autre exercice

Écrire une fonction `cartes(N)` qui renvoie une main de N cartes prises au hasard dans un jeu de 32 cartes. Par exemple, avec `cartes(2)`, on devrait pouvoir obtenir `[As de Trefle, Dix de Pique]`.

Une première approche est la fonction `cartes(n)` qui choisit n fois une valeur et une couleur dans les listes correspondantes avec la fonction `choice(liste)`.

```
from random import *
```

```
def cartes(n):
  couleur=['Trefle','Carreau','Coeur','Pique']
  valeur=['7','8','9','10','Valet','Dame','Roi','As']
  main=[]
  for i in range(n):
    carte="{} de {}".format(choice(valeur),choice(couleur))
    main.append(carte)
  return main
```

Pour éviter d'avoir deux cartes identiques, on peut remplacer la boucle **for** par une boucle **while**. La condition d'arrêt du **while** peut porter sur la longueur de la liste **main** créée. Avant d'ajouter une nouvelle carte, on vérifie que l'élément n'est pas déjà présent dans la liste.

```
from random import *
def cartes(n):
  couleur=['Trefle','Carreau','Coeur','Pique']
  valeur=['7','8','9','10','Valet','Dame','Roi','As']
  main=[]
  while len(main)<n:
    carte="{} de {}".format(choice(valeur),choice(couleur))
    if carte not in main:
      main.append(carte)
  return main
```

Pour prolonger ce travail, on pourrait écrire un programme qui renvoie P mains différentes de N cartes, les P mains ne pouvant contenir les mêmes cartes. On pourrait ainsi distribuer le jeu entre P joueurs, par exemple `cartes(8,4)` distribuerait 8 cartes à 4 joueurs.
Pour réaliser ce projet, il serait peut-être plus judicieux de générer une liste contenant l'ensemble des cartes et de tirer ensuite dans cette liste un élément au hasard que l'on supprimerait de la liste (pour éviter l'inconvénient d'une boucle **while** qui tire très souvent et pour rien des cartes déjà tirées). Le programme qui suit réalise cela, d'une part en parcourant les deux listes (**couleur** et **valeur**) en entier pour constituer la liste **jeu**, et d'autre part en tirant un élément de la liste jeu au hasard et en l'y supprimant avec cette instruction à tiroirs : `main.append(jeu.pop(randrange(len(jeu))))`.

```
from random import *
def cartes(n,p):
  couleur=['Trefle','Carreau','Coeur','Pique']
  valeur=['7','8','9','10','Valet','Dame','Roi','As']
  jeu,mains=[],[]
  for c in couleur:
    for v in valeur:
      jeu.append("{} de {}".format(v,c))
  for i in range(p):
```

```
    main=[]
    while len(main)<n:
      main.append(jeu.pop(randrange(len(jeu))))
      mains.append(main)
  return mains
```

Pour distribuer 8 cartes à chacun des 4 joueurs, on peut écrire :

```
nb_cartes,nb_joueurs=8,4
jeux=cartes(nb_cartes,nb_joueurs)
for i in range(nb_joueurs):
  print("Joueur {}:".format(i+1))
  print(" - ".join(jeux[i]))
```